UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
Y SISTEMAS DE TELECOMUNICACIÓN
Campus Sur. Ctra. de Valencia km. 7. 28031 Madrid

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Gestión del conocimiento en redes de sensores inalámbricos.

TÍTULO: Gestión de la información en redes de sensores inalámbricos.

AUTOR: Javier Vicente Vallejo

TUTOR: Rubén de Diego Martínez **Vº Bº**.

DEPARTAMENTO: Diatel

Miembros del tribunal calificador

PRESIDENTE: Francisco José Arqués Orobón
VOCAL: Rubén de Diego Martínez
VOCAL SECRETARIO: Vicente Hernández Díaz
DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

Wireless Sensor Networks (WSN) son redes inalámbricas formadas por un conjunto de sensores que miden algún tipo de parámetro del entorno, donde los datos recolectados son enviados al coordinador para su análisis.

Se va a realizar un estudio del estado del arte en cuanto a la gestión de la información en WSN, centrando dicho estudio en la plataforma TinyDB por ser ésta la más avanzada en cuanto a abstracción en el manejo de datos en el entorno que nos ocupa.

Posteriormente, se implementará un sistema real de monitorización de consumo energético para entornos industriales y domésticos basado en WSN. En dicho sistema se implementarán diversas optimizaciones inspiradas en la plataforma estudiada TinyDB.

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior de Ingeniería y

Sistemas de Telecomunicación

PROYECTO FIN DE CARRERA

Gestión de la información en redes de sensores inalámbricos

AUTOR

Javier Vicente Vallejo

TUTOR

Rubén De Diego Martínez

Julio de 2014

Resumen

La gestión del conocimiento (KM) es el proceso de recolectar datos en bruto para su análisis y filtrado, con la finalidad de obtener conocimiento útil a partir de dichos datos. En este proyecto se pretende hacer un estudio sobre la gestión de la información en las redes de sensores inalámbricos como inicio para sentar las bases para la gestión del conocimiento en las mismas.

Las redes de sensores inalámbricos (WSN) son redes compuestas por sensores (también conocidos como motas) distribuidos sobre un área, cuya misión es monitorizar una o varias condiciones físicas del entorno. Las redes de sensores inalámbricos se caracterizan por tener restricciones de consumo para los sensores que utilizan baterías, por su capacidad para adaptarse a cambios y ser escalables, y también por su habilidad para hacer frente a fallos en los sensores.

En este proyecto se hace un estudio sobre la gestión de la información en redes de sensores inalámbricos. Se comienza introduciendo algunos conceptos básicos: arquitectura, pila de protocolos, topologías de red, etc.… Después de esto, se ha enfocado el estudio hacia TinyDB, el cual puede ser considerado como parte de las tecnologías más avanzadas en el estado del arte de la gestión de la información en redes de sensores inalámbricos.

TinyDB es un sistema de procesamiento de consultas para extraer información de una red de sensores. Proporciona una interfaz similar a SQL y permite trabajar con consultas contra la red de sensores inalámbricos como si se tratara de una base de datos tradicional. Además, TinyDB implementa varias optimizaciones para manejar los datos eficientemente.

En este proyecto se describe también la implementación de una sencilla aplicación basada en redes de sensores inalámbricos. Las motas en la aplicación son capaces de medir la corriente a través de un cable. El objetivo de esta aplicación es monitorizar el consumo de energía en diferentes zonas de un área industrial o doméstico, utilizando redes de sensores inalámbricas. Además, se han implementado las optimizaciones más importantes que se han aprendido en el análisis de la plataforma TinyDB.

Para desarrollar esta aplicación se ha utilizado como sensores la plataforma *open-source* de creación de prototipos electrónicos Arduino, y el ordenador de placa reducida Raspberry Pi como coordinador.

Abstract

Knowledge management (KM) is the process of collecting raw data for analysis and filtering, to get a useful knowledge from this data. In this project the information management in wireless sensor networks is studied as starting point before knowledge management.

Wireless sensor networks (WSN) are networks which consists of sensors (also known as motes) distributed over an area, to monitor some physical conditions of the environment. Wireless sensor networks are characterized by power consumption constrains for sensors which are using batteries, by the ability to be adaptable to changes and to be scalable, and by the ability to cope sensor failures.

In this project it is studied information management in wireless sensor networks. The document starts introducing basic concepts: architecture, stack of protocols, network topology… After this, the study has been focused on TinyDB, which can be considered as part of the most advanced technologies in the state of the art of information management in wireless sensor networks.

TinyDB is a query processing system for extracting information from a network of sensors. It provides a SQL-like interface and it lets us to work with queries against the wireless sensor network like if it was a traditional database. In addition, TinyDB implements a lot of optimizations to manage data efficiently.

In this project, it is implemented a simple wireless sensor network application too. Application's motes are able to measure amperage through a cable. The target of the application is, by using a wireless sensor network and these sensors, to monitor energy consumption in different areas of a house. Additionally, it is implemented the most important optimizations that we have learned from the analysis of TinyDB platform.

To develop this application it is used Arduino open-source electronics prototyping platform as motes, and Raspberry Pi single-board computer as coordinator.

Índice del Documento

Resumen i
Abstract iii
Índice del Documento v
Índice de figuras vii
Índice de tablas ix
Acrónimos xi
Capítulo 1 Introducción y Objetivos 1
1.1. Objetivos 2
1.2. Estructura 2
Capítulo 2 Base teórica y estudio del estado del arte 5
2.1. Standard IEEE 802.15.4 5
2.1.1. Capa física IEEE 802.15.4 6
2.1.2. Capa MAC IEEE 802.15.4 7
2.1.2.1. Funciones soportadas por la capa MAC de IEEE 802.15.4 8
2.1.2.2. Modos de operación en IEEE 802.15.4 capa MAC 8
2.2. *Standard* ZigBee 9
2.2.1. *Standard* ZigBee: *Network layer* (NWK) 10
2.2.1.1. Dispositivos 11
2.2.1.2. Procedimiento de unión a la red 11
2.2.1.3. Asignación de direcciones 11
2.2.1.4. Enrutamiento en NWK de ZigBee 12
2.2.2. *Standard* ZigBee: *Application layer* 14
2.2.2.1. Objetos 14
2.2.2.2. Perfiles 15
2.2.2.3. Servicios 15
2.2.3. *Standard* ZigBee: Enrutamiento 16
2.2.4. *Standard* ZigBee: Localización de los dispositivos 18
2.3. Modelos de almacenamiento de datos en WSN 18
2.4. TinyDB 19
2.4.1. Modelos de datos en TinyDB 20
2.4.2. Tipos de *queries* en TinyDB 21
2.4.3. Sintaxis de las *queries* en TinyDB 22
2.4.4. Bases de la gestión de la red en TinyDB 25
2.4.5. Estrategias de optimización de TinyDB 26
2.4.5.1. Estrategias de optimización de TinyDB basadas en el uso de la red 26
Manejo de metadatos 26
Snooping 27
Semantic routing trees (SRTs) 28
2.4.5.2. Estrategias de optimización de TinyDB locales en cada nodo 29
Política de ejecución de *queries*, EPOCHs 29
Política de ejecución de *queries*: *queries* múltiples 31
Política de ejecución de *queries*: priorización de entrega de datos 31
Lifetime – adaptación de ratios para adaptar consumo de energía 32
Capítulo 3 Especificación del problema a resolver 35
3.1. Aplicación para monitorización de consumo energético 35
3.2. Caso de uso centro de control 37

3.3. Caso de uso red de sensores 38
3.4. Márgenes de error y optimizaciones 39
Capítulo 4 Diseño de la aplicación 41
4.1. Elección del hardware a utilizar 41
4.1.1. Elección del hardware para los nodos de la WSN 41
4.1.2. Elección del sensor de corriente eléctrica 43
4.1.3. Elección del hardware para comunicaciones ZigBee 44
4.1.4. Elección del hardware para el centro de control 47
4.2. Elección del entorno de desarrollo a utilizar 51
4.2.1. Elección del entorno de desarrollo para el gestor de los nodos de la red de sensores 52
4.2.2. Elección del entorno de desarrollo para el gestor del centro de control 53
4.3. Diseño del protocolo de comunicación 53
4.3.1. Caso de uso de los nodos medidores 53
4.3.2. Solicitudes y respuestas 54
4.3.2.1. Recibir medición 54
4.3.2.2. Configurar umbral mínimo para envío de datos 55
4.3.2.3. Notificación batería baja 56
4.4. Diseño de la aplicación para el coordinador de la red de sensores 56
4.4.1. Diagrama de flujo coordinador de la red de sensores 57
4.5. Diseño de la aplicación para cada nodo medidor 57
4.5.1. Optimizaciones en la aplicación de los nodos medidores 58
4.5.1.1. EPOCH (*once per time-period*) 58
4.5.1.2. Almacenamiento de datos en la red 59
4.5.1.3. Esquema de pila *naive* 59
4.5.1.4. Filtrado de datos realizado directamente en los nodos 59
4.5.1.5. *Lifetime* 60
4.5.2. Diagrama de flujo de la aplicación de los nodos medidores 60
4.6. Diseño de la aplicación para el centro de control principal 62
4.6.1. Diagrama de clases del centro de control 65
Capítulo 5 Implementación de la aplicación 67
5.1. Detalles de implementación en el coordinador de la red de sensores 67
5.1.1. Configuración del módulo Xbee Series 2 como coordinador 67
5.1.2. Solución al conflicto del puerto serie: USB y Xbee Series 2 71
5.1.3. Código en java para el Arduino coordinador 72
5.2. Detalles de implementación en la aplicación de los nodos medidores 72
5.2.1. Configuración del módulo Xbee Series 2 como dispositivo final 72
5.2.2. Detalles de implementación del "*Once Per-Time Period*" EPOCH 73
5.2.3. Detalles de implementación del *lifetime* 77
5.2.4. Calibración del API de lectura de Emontx 78
5.3. Detalles de implementación en el centro de control principal 81
Capítulo 6 Pruebas realizadas 83
6.1. Pruebas durante el desarrollo del medidor 83
6.2. Pruebas durante el desarrollo del coordinador 84
6.3. Pruebas durante el desarrollo del centro de control 85
6.4. Pruebas del sistema completo 85
Capítulo 7 Conclusiones 91
7.1 Trabajos futuros 92
Capítulo 8 Bibliografía 93

Índice de figuras

Figura 1. Pila de protocolos IEEE 802.15.4 6
Figura 2: topologías de red IEEE 802.15.4 capa MAC 7
Figura 3: superframes de la capa MAC IEEE 802.15.4 8
Figura 4: pila de protocolos ZigBee 10
Figura 5: ejemplo de asignación de direcciones en la capa NWK de ZigBee 12
Figura 6: procesamiento de RREQ en NWK de ZigBee con topología de malla 13
Figura 7: procesamiento de RREP en NWK de ZigBee con topología de malla 14
Figura 8: enrutamiento en ZigBee, problema agujeros en la red 17
Figura 9: enrutamiento perimetral en ZigBee 17
Figura 10: ejemplo de utilización de árboles semánticos en TinyDB 29
Figura 11: ejemplo de utilización de EPOCHs en TinyDB 31
Figura 12: esquema del sistema de monitorización de consumo 36
Figura 13: caso de uso centro de control 38
Figura 14: caso de uso elemento medidor 39
Figura 15: placa base Arduino 42
Figura 16: sensor SCT-013 43
Figura 17: placa Emontx 44
Figura 18: módulo Xbee series 2 45
Figura 19: Xbee Shield Pro 46
Figura 20: Conexión módulo Xbee series 2 con Xbee Shield Pro 46
Figura 21: Elementos del sistema y sus conexiones 47
Figura 22: Raspberry Pi 49
Figura 23: Raspberry Pi diagrama de bloques 50
Figura 24: Configuración hardware del sistema 51
Figura 25: IDE Arduino-1.0.5 52
Figura 26: Caso de uso nodos medidores 54
Figura 27: Petición/respuesta recibir medición 55
Figura 28: Petición/respuesta configuración de umbral mínimo 55
Figura 29: Petición/respuesta notificación batería baja 56
Figura 30: Diagrama de flujo del coordinador de la red ZigBee 57
Figura 31: Diagrama de flujo principal medidor 61
Figura 32: Diagrama de flujo medidor gestión de comandos 62
Figura 33: Centro de control opciones 63
Figura 34: Centro de control muestra de datos recibidos 64
Figura 35: Centro de control muestra de datos recibidos 64
Figura 36: Diagrama de clases del centro de control 65
Figura 37: Configuración conexión X-CTU 68
Figura 38: Número de serie Xbee Series 2 con X-CTU 69
Figura 39: Configuración del *modem* del coordinador *Zigbee* 70
Figura 40: Configuración del *modem* de los dispositivos finales Zigbee 73
Figura 41: Utilidad monitor serie del IDE Arduino-1.0.5 84
Figura 42: Mediciones utilizando bombilla de 25W con dos nodos en la red 86
Figura 43: Mediciones utilizando bombilla de 40W con dos nodos en la red 87
Figura 44: Mediciones utilizando bombilla de 60W con dos nodos en la red 88
Figura 45: Mediciones utilizando bombilla de 100W con dos nodos en la red 89

Índice de tablas

Tabla 1: estructura filas tabla sensors de TinyDB 20
Tabla 2: información en el catálogo de metadatos 21
Tabla 3: sintaxis de las queries de TinyDB 24
Tabla 4: modos *standby* de Arduino 74
Tabla 5: tabla de valores configurables para watchdog timer 75
Tabla 6: configuración watchdog timer (código) 76
Tabla 7: entrada de Arduino a modo standby (código) 76
Tabla 8: Arduino lectura de voltaje de la batería (código) 77
Tabla 9: Emontx calibración (código) 79
Tabla 10: Emontx calibración, mediciones para constante 111.1 80
Tabla 11: Emontx calibración, mediciones para constante 29.1 80
Tabla 12: Emontx calibración, mediciones para constante 56.9 81

Acrónimos

WSN: Wireless Sensors Network
KM: Knowledge Management
IEEE: Institute of Electrical and Electronics Engineers
SQL: Structured Query Language
NWK: Network Layer
MAC: Media Access Control
DSSS: Direct Seq Spread Spectrum
RFD: Reduced Function Device
FFD: Full Function Device
Q-PSK: Quadrature Phase Shift Keying
B-PSK: Binary Phase Shift Keying
AODV: Ad-Hoc on Demand Distance Vector
RDT: Routing Discovery Table
GPSR: Greedy Perimeter Stateless Routing
DNS: Domain Name Service
TCP/IP: Transmission Control Protocol/Internet Protocol
APO: Application Object
ZDO: Zigbee Device Object
GPS: Global Positioning System
RSSI: Received Signal Strength Indicator
TDoA: Time Difference Of Arrival
AoA: Angle Of Arrival
SRT: Semantic Routing Tree
FIFO: First In First Out
EPOCH: Once Per Time-Period
API: Application Interface
USB: Universal Serial Bus
PAN ID: Personal Area Network Identifier
FTDI: Future Technology Devices International

Capítulo 1 Introducción y Objetivos

El mundo de las redes de sensores inalámbricos (WSN) es un mundo que actualmente se encuentra en plena evolución. Una red de sensores inalámbricos consiste de una red de pequeñísimos ordenadores, las motas, equipados con sensores, que colaboran en una tarea común.

La finalidad de los nodos o motas de la red de sensores inalámbricos en recopilar cierta información física del entorno en el que se encuentran desplegadas, por lo que en este tipo de redes suele existir un gran flujo de datos. Datos en bruto que deben ser filtrados, clasificados, organizados,… para obtener un conocimiento útil.
[1]

Es por ello que se debe estudiar a fondo y dar importancia a la gestión del conocimiento (KM) en el mundo de las redes de sensores inalámbricos. Buscar la mejor manera de gestionar la información de que se dispone en la red para obtener conocimiento útil y a la vez utilizar la red de manera óptima para maximizar el tiempo de vida de las motas y minimizar el tráfico en la red con datos poco útiles o innecesarios.
[2]

En este trabajo se va a realizar un estudio del estado del arte de la gestión de la información en las redes de sensores inalámbricos, como punto de inicio para en futuros trabajos estudiar la gestión de conocimiento. Se estudiará en profundidad la plataforma TinyDB como ejemplo de estado del arte en este campo, y se estudiará las optimizaciones en cuanto a manejo de datos que implementa dicha plataforma.

A continuación se implementará una aplicación real construida sobre una red de sensores inalámbricos, cuya finalidad será monitorizar el consumo de energía en diferentes zonas de una casa. Para ello se utilizarán motas basadas en la placa Arduino, dotadas de módulos de comunicación ZigBee y sensores de corriente en cables eléctricos.
[3]
Se implementará además una interfaz de usuario utilizando un ordenador de placa reducida Raspberry Pi.
[4]

1.1. Objetivos

En este proyecto se ha tratado de cubrir los siguientes objetivos:

- Realizar un estudio del estado del arte en gestión de la información en redes de sensores inalámbricos.
- Realizar un estudio de la plataforma TinyDB y las diferentes optimizaciones que implementa para manejo de datos, con el fin de reducir el tráfico en la red de sensores así como maximizar el tiempo de vida de las motas.
- Utilizar la plataforma *open-source* de creación de prototipos electrónicos Arduino para el desarrollo de los sensores de una red de sensores inalámbricos.
- Utilizar el ordenador de placa reducida Raspberry Pi para el desarrollo del coordinador de una red de sensores inalámbricos.
- Implementar una aplicación en el ámbito del hogar inteligente y el ahorro energético que, basada en una red de sensores inalámbricos, sea capaz de monitorizar el consumo energético en diferentes zonas de una casa.
- Configurar cada módulo ZigBee en cada dispositivo final así como en el coordinador, para permitir la comunicación entre éste y los dispositivos finales.
- Implementar la funcionalidad necesaria en cada mota de la red para realizar mediciones de la corriente a través de un cable, utilizando sensores de tipo TA12-200.
- Implementar la funcionalidad necesaria para comunicar cada mota con el coordinador y realizar el envío de las mediciones realizadas.
- Implementar las optimizaciones más importantes aprendidas en el estudio de la plataforma TinyDB en nuestra aplicación de monitorización de consumo energético con el fin de minimizar el tráfico en la red y aumentar el tiempo de vida de nuestras motas.
- Implementar la funcionalidad necesaria para poner en comunicación el módulo Arduino del coordinador con la placa Raspberry Pi a través de USB, para poder enviar las mediciones realizadas al ordenador Raspberry.
- Implementar una interfaz de usuario sobre Raspberry Pi que permita al usuario de la red acceder a las mediciones realizadas.

1.2. Estructura

Este documento estará dividido en ocho capítulos:

- Introducción, objetivos y estructura del documento.

En el primer capítulo se hace una introducción para posteriormente definir los objetivos de este trabajo y por último concretar la estructura de este documento.

- Base teórica y estudio del estado del arte.

  A continuación, en el segundo capítulo, se comienza haciendo un resumen, una base teórica en cuanto a arquitectura, protocolos, topologías de red, etc.… de las redes de sensores inalámbricos, para a continuación, hacer un estudio del estado del arte en lo referido a gestión de la información en dichas redes.

- Especificación del problema a resolver.

  En el siguiente capítulo, capítulo 3, se especifica el problema que se pretende resolver con la aplicación basada en redes de sensores inalámbricos que se va a desarrollar.

- Diseño.

  En el capítulo 4 se entra más a fondo a la parte más técnica del trabajo, es dónde se define el diseño de la aplicación.

- Implementación.

  En el apartado de implementación, capitulo 5, se entrará de lleno en el desarrollo del sistema comentando detalles destacados de la implementación.

- Pruebas.

  Posteriormente en el capítulo 6 se realizan varias pruebas sobre dicha aplicación, asegurando su correcto funcionamiento.

- Conclusiones.

  Para finalizar, el último capítulo constará de una serie de conclusiones sacadas tras la elaboración de este trabajo así como la implementación de la aplicación basada en redes de sensores inalámbricos que se van a desarrollar.

El documento terminará con posibles trabajos futuros en el ámbito que se abarca, así como una serie de referencias bibliográficas consultadas tanto para realizar la parte teórica del proyecto como para la parte práctica.

Capítulo 2 Base teórica y estudio del estado del arte

En este capítulo se va a hacer un estudio teórico de los protocolos que intervienen en las redes de sensores inalámbricos. Se hará un breve repaso al estándar IEEE 802.15.4 utilizado en el nivel físico de este tipo de redes, que además es la base sobre la que se define la especificación ZigBee. Se continuará haciendo un pequeño estudio de esta especificación, que ofrece una solución completa para redes de sensores inalámbricos, aportando los niveles superiores de la pila de protocolos, no cubiertos por IEEE 802.15.4.

Tras esta introducción teórica de los protocolos fundamentales de las redes de sensores inalámbricos, se quiere profundizar en el sistema de procesamiento de *queries* TinyDB, construido sobre ZigBee. TinyDB permite gestionar una red de sensores inalámbricos tratando la misma como si fuera una base de datos y realizando consultas similares a SQL sobre la misma red. Se podría considerar a TinyDB el estado del arte en cuanto a gestión de la información en las redes de sensores inalámbricos. Por otro lado, aunque en la aplicación que se desarrollará no se usará TinyDB, por no ser soportado por la plataforma que se va a utilizar, interesa realizar un estudio de las optimizaciones que implementa TinyDB para recolectar eficientemente los datos de la red de sensores. En la aplicación que se va a desarrollar, se implementarán algunas de estas optimizaciones, adaptadas a la plataforma usada, las optimizaciones que más interesan para los objetivos de la misma.

2.1. Standard IEEE 802.15.4

El estándar IEEE 802.15.4 define el nivel físico y el control de acceso al medio de las LR-WPAN: *low-rate wireless personal area network* (redes inalámbricas de área personal con tasas bajas de transmisión de datos).

Se compone de dos capas, la capa física y la capa MAC.
[5]

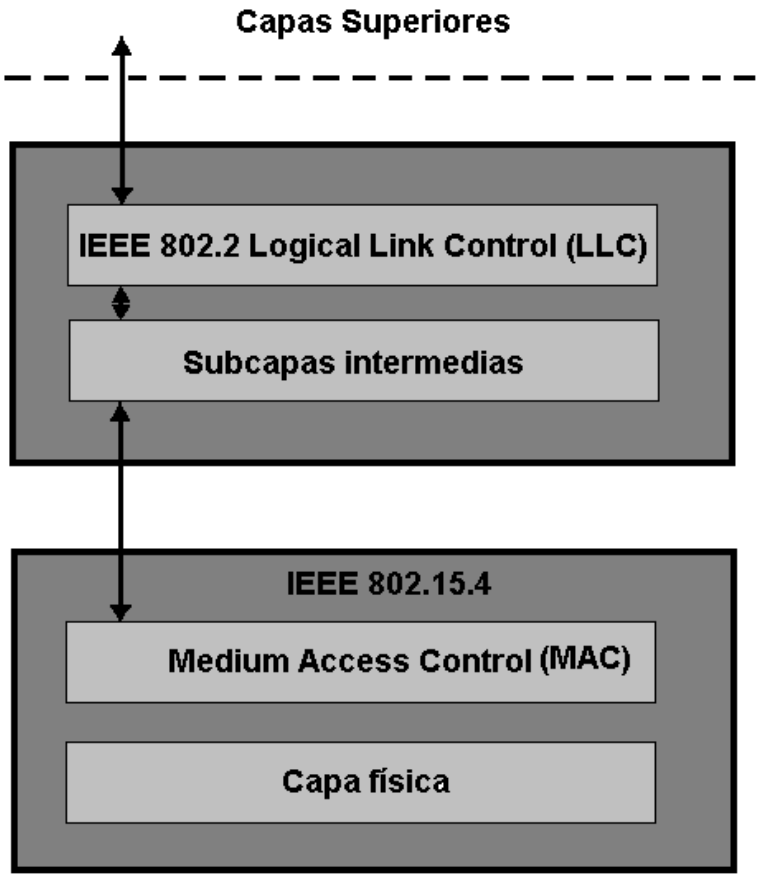

Figura 1. Pila de protocolos IEEE 802.15.4

2.1.1. Capa física IEEE 802.15.4

La capa física soporta las funciones de selección de canal, estimación de calidad de enlace, medición de detección de energía y evaluación de canal libre para ayudar a la selección de canal. Utiliza un modo de acceso de tipo DSSS (*Direct Seq Spread Spectrum*) y alguna de las siguientes bandas
[6]:

- 2450 MHz (16 Ch) Modulación 0-QPSK.
- 915 MHz (10 Ch) Modulación BPSK.
- 868 MHz (1 Ch) Modulación BPSK.

2.1.2. Capa MAC IEEE 802.15.4

La capa de control de acceso al medio (MAC) utiliza el canal físico para transmitir tramas MAC. Ofrece servicio de transmisión de datos y de control, y regula el acceso al canal físico. Regula la validación de las tramas, gestiona la asociación entre nodos, garantiza *slots* de tiempo y ofrece puntos de enganche para servicios seguros.

En la capa MAC del estándar IEEE 802.15.4 se tienen dos tipos de nodos, los RFD, que son los que actúan como dispositivos finales, y los FFD, que suelen actuar como *routers*. Los RFD interactúan con un único FFD. Los FFD ofrecen sincronización mediante envío de un *bacon* y servicios de unión a la red.

Los nodos pueden adquirir diferentes roles. Los FFD pueden actuar como coordinador, en cuyo caso tienen capacidad para hacer de *router* para otros FFD o para RFD. También pueden adquirir el rol de coordinador PAN. En este caso el FFD es coordinador, pero además es el coordinador de toda la red de área personal, la PAN. Los RFD, que no actúan como *router*, sólo pueden adquirir el rol de dispositivo final.

En cuanto a la topología de red se pueden encontrar topologías en forma de estrella o topologías de tipo peer-2-peer.
[6]

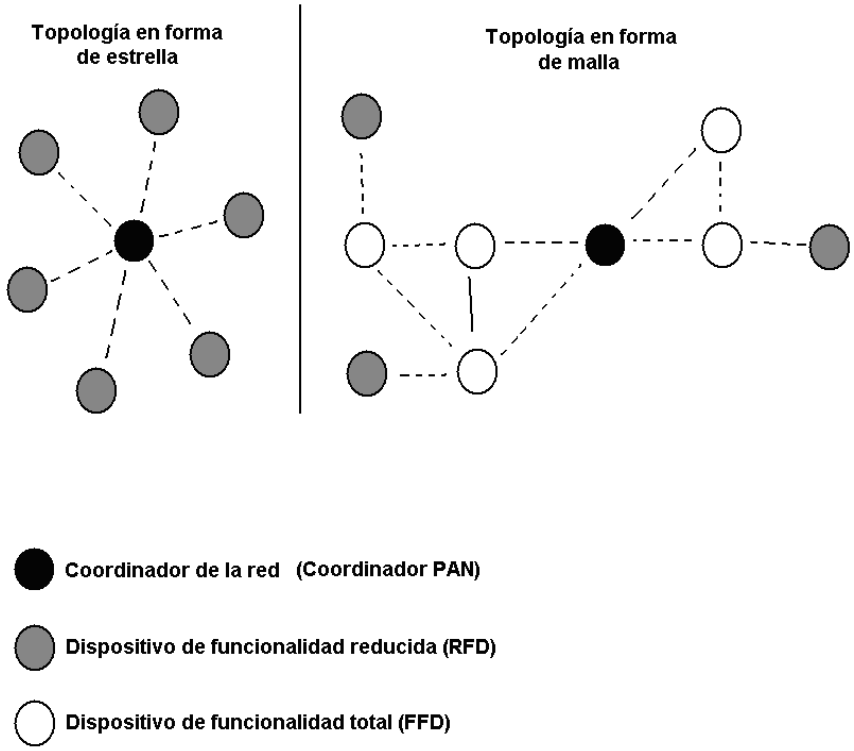

Figura 2: topologías de red IEEE 802.15.4 capa MAC

2.1.2.1. Funciones soportadas por la capa MAC de IEEE 802.15.4

La capa MAC soporta las siguientes funciones:

- Transferencia de datos.
- Funcionalidad para asociación / disociación de nuevos nodos. Mediante un sistema de registro en la red el nodo recibe una dirección de 16 bits.
- Escaneo de canal:
- Escaneo activo: Los FFD envían un *bacon* y se quedan a la escucha de respuestas. De esta manera los FFD localizan RFD.

Escaneo pasivo: Sólo se espera el bacon. Lo usan los RFD para localizar FFD (PAN y coordinadores).
[6]

2.1.2.2. Modos de operación en IEEE 802.15.4 capa MAC

En IEEE 802.15.4 se pueden encontrar los siguientes modos de operación en su capa MAC:

Con *superframes*:

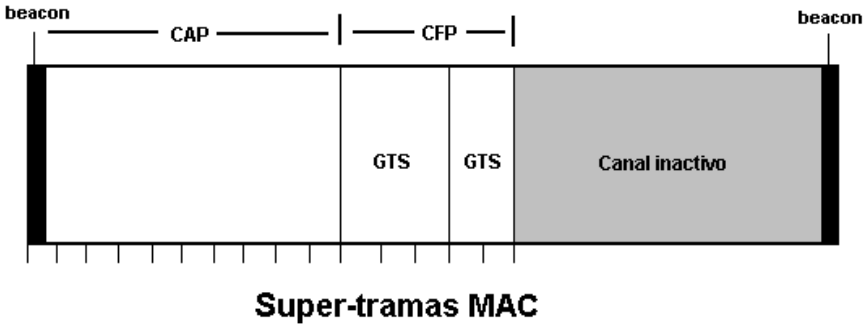

Figura 3: superframes de la capa MAC IEEE 802.15.4

- Comunicación coordinador a dispositivo final: cada cierto tiempo el dispositivo final espera el *bacon*. Cuando el coordinador tiene algo para el dispositivo final, marca el

bacon con el *flag* de "mensaje en espera", así el dispositivo final sabe que debe intentar recuperar el mensaje.

- Comunicación dispositivo final a coordinador: el dispositivo final espera el *bacon* para sincronizarse. Luego compite por un canal con CSMA-CA.

Sin *superframes*:

- No hay *bacon*.
- Comunicaciones mediante *unslotted* CSMA-CA.
- La comunicación entre dispositivos finales y coordinador se realiza sin problemas porque el coordinador siempre está encendido y listo para recibir.
- La comunicación coordinador y dispositivos finales se realiza mediante *polling*.
- La comunicación coordinador y coordinador no tiene problemas ya que todos están encendidos.

[6]

2.2. *Standard* ZigBee

En Standard ZigBee define un conjunto de protocolos de alto nivel para su utilización en comunicaciones inalámbricas de radiodifusión digital de bajo consumo. Se basa en el estándar IEEE 802.15.4, y está orientado a aplicaciones distribuidas que requieren una tasa de envío de datos baja, pero que necesitan minimizar el consumo.
[7]

Aquí se puede ver un esquema de la pila de protocolos definidos en esta especificación:

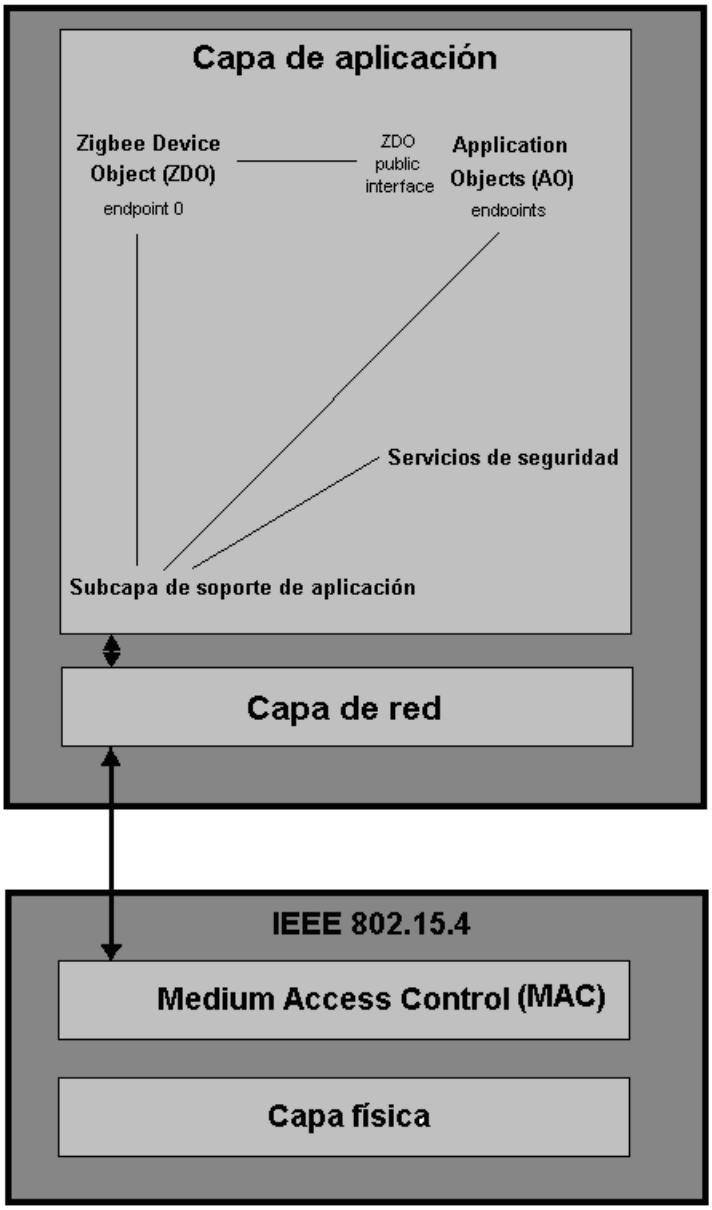

Figura 4: pila de protocolos ZigBee

2.2.1. *Standard* ZigBee: *Network layer* (NWK)

En este apartado se va a hacer una introducción a la capa de red del estándar ZigBee. A continuación se va a hablar del tipo de dispositivos en este tipo de redes, de cómo se unen a la red y cómo se asignan direcciones a los dispositivos, y de cómo ocurre el enrutamiento en las redes ZigBee.

2.2.1.1. Dispositivos

Se tienen tres tipos de dispositivos en la capa NWK de las redes ZigBee, muy similares a los de IEEE 802.15.4:

- Dispositivo final ZigBee. Es un RFD o FFD actuando como nodo simple, sin capacidad de enrutamiento.
- Router ZigBee. Es un FFD con capacidad de enrutamiento.

Coordinador ZigBee. Es un FFD que gestiona toda la red ZigBee.
[7]

2.2.1.2. Procedimiento de unión a la red

A continuación se describe el procedimiento de unión a la red en la capa NWK de ZigBee.

Se tendría por un lado un dispositivo C que desea unirse a la red y hace una petición a la capa NWK para empezar un procedimiento de descubrimiento. NWK utiliza la capa MAC para aprender sobre su vecindario y los routers de su vecindario. Después de realizar este proceso de descubrimiento, la capa NWK elige un router (un padre) P de su vecindario.

A continuación la capa NWK pide a la capa MAC que inicie un procedimiento de asociación de C con P.

Ya en P (el nodo de la red que actuará como padre de C), la capa NWK recibe una indicación de petición de asociación desde la capa MAC. La capa NWK en P asigna una dirección de red de 16 bits para el nuevo nodo y pide a la capa MAC que responda a la petición de asociación. El nodo C usará la dirección de 16 bits asignada para las siguientes comunicaciones.

De esta manera se ha creado una relación padre-hijo entre P y C. C pasa a ser un nodo en la estructura de árbol de la red, en la cual el coordinador ZigBee es la raíz de dicho árbol.
[8]

2.2.1.3. Asignación de direcciones

Se verá a continuación cómo se realiza la asignación de direcciones en la capa NWK de ZigBee.

Se juega con un conjunto de parámetros que son seleccionados por el coordinador de la red ZigBee:

- Rm: máximo número de *routers* que pueden ser hijos de otro *router*.
- Dm: máximo número de dispositivos finales que pueden ser hijos de otro *router*.
- Lm: máxima profundidad del árbol.

El proceso de asignación es el siguiente. Cuando se une un nuevo *router* a la red se le asigna un rango de direcciones de 16 bits consecutivas. El primer entero en el rango es la dirección del propio nodo. El resto están disponibles para que dicho nodo se las asigne a sus hijos en el árbol (tanto *routers* como dispositivos finales).

A continuación se puede ver un ejemplo gráfico de este proceso:

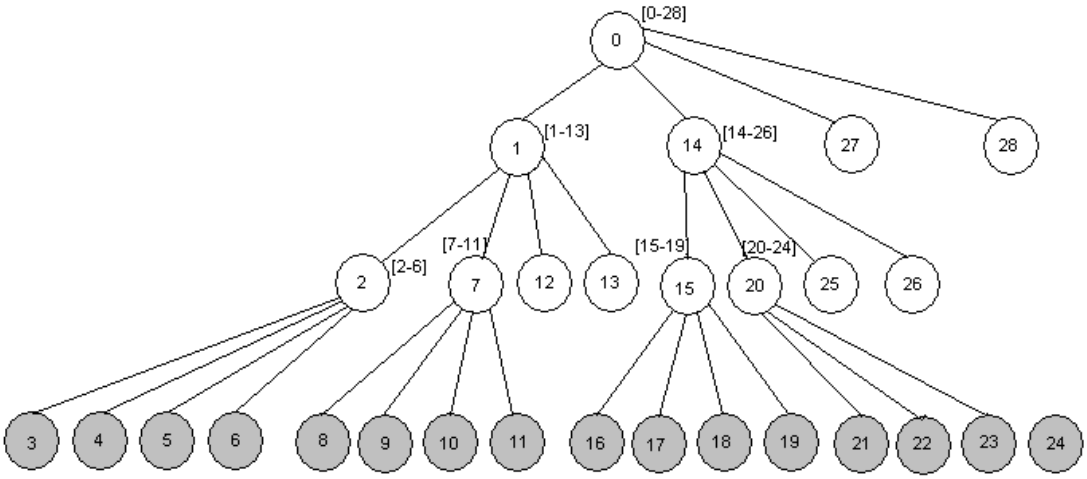

Figura 5: ejemplo de asignación de direcciones en la capa NWK de ZigBee

En el ejemplo de la figura se puede ver como el nodo raíz dispone de las direcciones de 0 a 28. Le asigna la 27 y la 28 a dos dispositivos finales cuyo padre es él mismo. Y le asigna direcciones de 14 a 26 al nodo 14 para que pueda asignárselas a sus hijos, y las direcciones de 1 a 13 al nodo 1. El resto de los nodos del árbol de la red actúan de la misma manera.
[8]

2.2.1.4. Enrutamiento en NWK de ZigBee

Se pueden encontrar dos tipos de topologías en ZigBee, en árbol y en malla.

En el caso de la topología en árbol el enrutamiento es más sencillo. Sólo existirán enlaces padre-hijo y cada *router* mantendrá su propia dirección y la de sus hijos. Mantendrán rangos de direcciones de cada rama que cuelga de ellos, y con esa información les será suficiente para realizar el enrutamiento.

La ventaja de esta topología en forma de árbol es que es simple y que permite el uso de *bacon*.

Por otro lado está la topología en forma de malla. En este caso existe un enrutamiento basado en AODV (*Adhoc on Demand Distance Vector Routing Algorithm*). Los *routers* mantienen una tabla RDT (*Routing Discovery Table*). Cuando un nodo necesita conocer un destino, emite un mensaje RREQ que se propaga (*broadcast*) por toda la red hasta alcanzar su destino. Cuando un nodo recibe un RREQ que va dirigido a él mismo, responde con un RREP. Mientras el RREP vuelve a su origen, tanto el origen como los nodos del camino que sigue el RREP y que pueden verlo, actualizan sus entradas de la RDT en el caso de que el coste sea más bajo que el coste actual de llegar a ese destino apuntado actualmente en su RDT.

A continuación se pueden ver los algoritmos de procesamiento de RREQ y RREP.
[8]

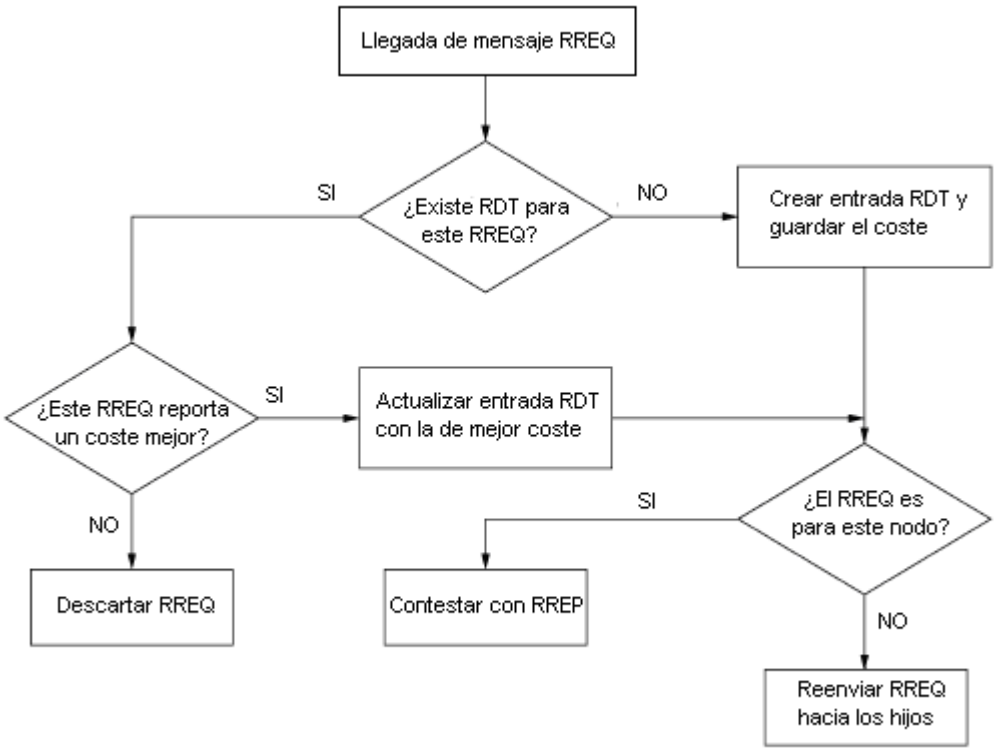

Figura 6: procesamiento de RREQ en NWK de ZigBee con topología de malla

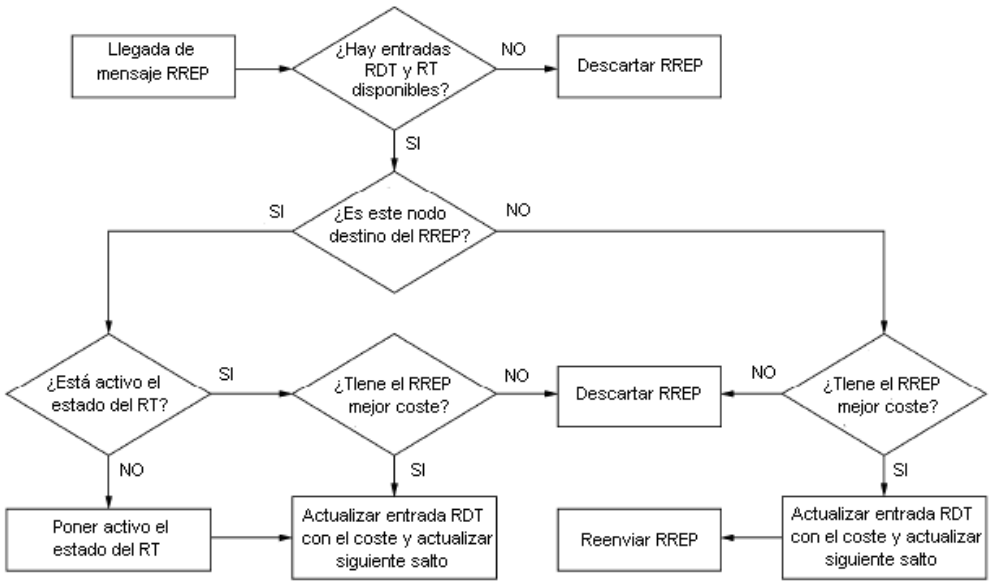

Figura 7: procesamiento de RREP en NWK de ZigBee con topología de malla

2.2.2. *Standard* ZigBee: *Application layer*

El servicio principal que ofrece la capa de aplicación es la transferencia de datos entre los objetos de esta capa, por ello se va a hablar primero de dichos objetos para luego entender cómo se relacionan dichos objetos y los servicios que ofrece la capa de aplicación.
[7]

2.2.2.1. Objetos

Application Object (APO): es una pieza de software que controla un elemento de hardware en un dispositivo de la red. Cada APO tiene asociado un *EndPoint Number*, que identifica dicho APO en un dispositivo para poder interactuar con él. Cada APO proporciona una serie de servicios y atributos para ser usados.
[8]

ZigBee Device Object (ZDO): es un tipo de objeto especial que ofrece servicios para ser usados por los APO. Estos servicios son:

- Descubrimiento de dispositivos.
- Descubrimiento de servicios implementados en cada dispositivo.
- Gestión de la comunicación.
- Gestión de la red.
- Gestión de seguridad.

2.2.2.2. Perfiles

Los perfiles de aplicación definen formatos de mensajes y protocolos para interactuar entre APO (esto permite a diferentes desarrolladores desarrollar independientemente, y construir y vender dispositivos ZigBee).

Aquí es importante hablar del término clúster. Un clúster es un conjunto de atributos de un APO desde el punto de vista de los perfiles. Se identifican con un id, y representa la interfaz (o parte de ella) que ofrece un APO hacia el exterior.

Por otro lado el perfil de dispositivo es un perfil especial que debe ser implementado por todos los dispositivos en la red ZigBee (ZDO), ya que da soporte para el descubrimiento de dispositivos y el descubrimiento de servicios.

Hay dos tipos de procedimientos de descubrimiento. Por un lado el modo de direccionamiento directo, en el que se realiza el envío de datos a una dirección de destino de dispositivo junto con un número de *endpoint*. En este caso el que envía debe realizar el *discovery*. Por otro lado está el modo de direccionamiento indirecto, donde el que envía sólo proporciona un id de cluster y necesita el soporte de un router o coordinador vecino para localizar el nodo destino del mensaje. Esto se puede hacer gracias a que la APS de los routers mantiene una tabla de asociaciones de este tipo: (src addr, src endp, cluster id) g (dst addr, dst endp).
[8]

2.2.2.3. Servicios

Los tipos de servicios de comunicación ofrecidos por la capa de aplicación son por un lado el tipo de servicio para clave-valor (KVP), en el que ZigBee define una plantilla de mensaje y los APO rellenan dichas plantillas para solicitar una operación sobre los atributos que residen en otro APO, y por otro lado el tipo de servicio de mensaje genérico, para aplicaciones que no se adaptan a los KVP.

[8]

Este último caso se deja la responsabilidad al perfil de aplicación de especificar los tipos de mensajes y sus contenidos.

2.2.3. *Standard* ZigBee: Enrutamiento

A continuación se va a hacer un breve repaso sobre las técnicas de enrutamiento que se utilizan en las redes ZigBee.
[8]

- *Adhoc On Demand Distance Vector* (AODV): es propuesto por ZigBee para las topologías en forma de malla. Es bastante ineficiente.
- Enrutamiento basado en árbol: los sensores recolectan datos y los envían hacia el "*sink*" (la raíz del árbol). Los sensores solo necesitan conocer a su nodo padre para poder enviar. No se soportan comunicaciones punto a punto entre dos dispositivos de la red. Este enrutamiento es posiblemente es más usado y el que se encontrará cuando se analice TinyDB.
- Enrutamiento geográfico: a los nodos se les asigna una localización, que puede ser una posición geográfica real o unas coordenadas virtuales. Los nodos difunden periódicamente su dirección a los vecinos y enrutan a través del vecino que minimiza la distancia al destino.
- Existe un problema con este tipo de enrutamiento, que es la posibilidad de que, debido a agujeros en la red, el encaminamiento se quede atrapado en un bucle intentando llegar al nodo destino (ya que siempre se encamina a través del nodo que minimiza la distancia al destino):

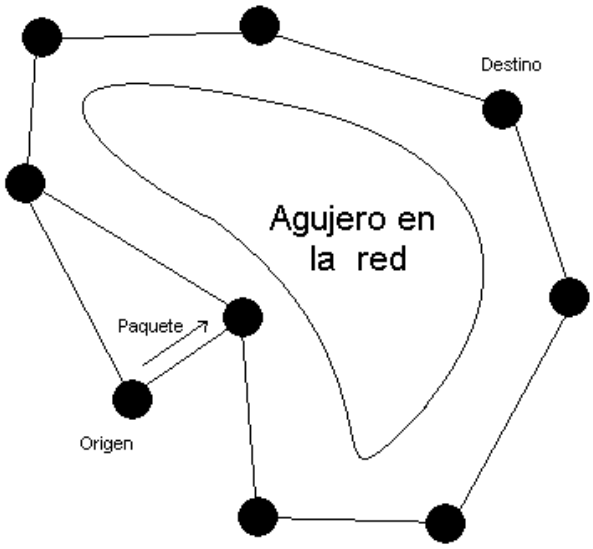

Figura 8: enrutamiento en ZigBee, problema agujeros en la red

La solución que se toma es realizar un encaminamiento perimetral para bordear el agujero:

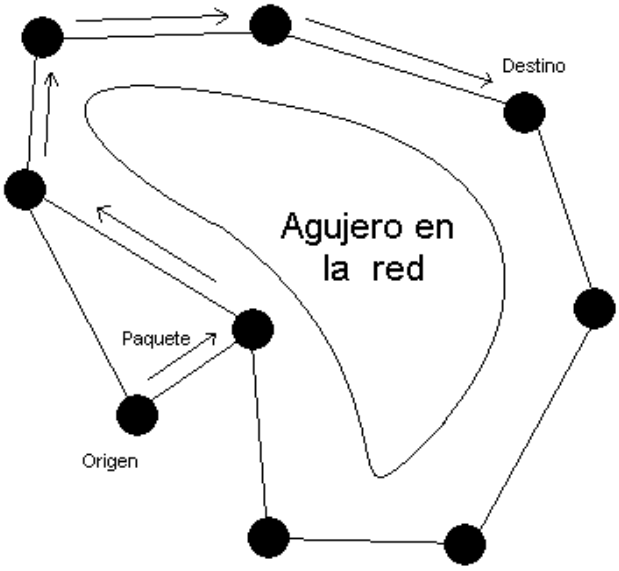

Figura 9: enrutamiento perimetral en ZigBee

2.2.4. *Standard* ZigBee: Localización de los dispositivos

Existen varias alternativas para geolocalizar a los dispositivos de la red. A continuación se verán algunas de ellas.
[8]

- GPS: Puede ocurrir que todos los nodos incorporen un GPS, o que solamente algunos nodos incorporen un GPS (los de los bordes) y los otros nodos utilicen métodos basados en mediciones de la señal de radio para saber su distancia relativa a nodos con GPS. Para realizar esta medición se miden los parámetros: fuerza de la señal recibida (RSSI), diferencia de tiempo de llegada (TDoA) y ángulo de la señal recibida (AoA).

- Coordenadas virtuales: Primero los nodos periféricos aprenden que ellos son nodos periféricos (mediante mediciones de radio y otras técnicas). Entonces los nodos periféricos inundan la red con mensajes "HELLO" para que todos los nodos periféricos sepan la distancia a los demás nodos periféricos. Finalmente cada nodo periférico computa su coordenada virtual mediante triangulación. Los nodos no periféricos computan su coordenada virtual mediante un algoritmo y a partir de las coordenadas de los periféricos.

- Servicio de localización: En este caso existen unos dispositivos especiales que convierten nombres a direcciones. Los nombres de cada nodo pueden ser basados en la función de dicho nodo (detector de luz, recolector de datos, líder de grupo 1, etc…), o en la posición geográfica (calle principal, cuadrante este, etc…). Es algo similar a DNS en las redes TCP/IP.

2.3. Modelos de almacenamiento de datos en WSN

Existen tres modelos principales de almacenamiento de datos en redes de sensores inalámbricos, que son los que se describen a continuación.
[8]

- **Almacenamiento externo**. Los datos se envían a un disco externo donde el centro de control realiza las *queries*.

- **Almacenamiento local**. Los nodos almacenan los datos que recopilan y las *queries* son inundadas en la red, y los nodos que tienen el dato contestan.

- ***Data Centric Storage*.** Después que se captura una medición, los datos se almacenan en algún nodo de la red y las queries se dirigen al nodo correcto. Se basa en una versión modificada de GPSR. La red entera es vista como una única hash table donde todos los nodos pueden usar dos operaciones:

    - PUT(key, value)
    - GET(key, value)

  A nivel de aplicación no se sabe dónde está almacenado el dato. Una *key* es mapeada a una coordenada geográfica.

  Es una buena aproximación cuando hay un gran número de nodos o muchos eventos se van a detectar (se van a medir muchos datos) pero no todos van a ser enviados al centro de control (ya que no llegarán queries preguntando por ellos).

  Un caso concreto de *Data Centric Storage*, es el *Data Centric Storage* orientado a *Ranged Queries*. En este caso los datos adyascentes se ubican en nodos cercanos. Los datos se convierten en coordenadas geográficas mediante una función de hash. Cuando un nodo genera una tupla de valores (lecturas de sus sensores), transforma la tupla en un código de zona. La función de *hashing* tiene la propiedad de que las tuplas con valores cercanos se almacenan en hojas adyascente del árbol (opcionalmente se puede implementar tolerancia a fallos duplicando los datos destinados a la zona Z en una zona *backup*(Z)).

2.4. TinyDB

La plataforma TinyDB es un sistema de procesamiento de *queries* para extracción de datos de una red de sensores inalámbricos. Proporciona una interfaz similar a SQL y permite trabajar con la red de sensores inalámbricos mediante *queries*. Desde el punto de vista del usuario, la red será vista como una base de datos tradicional, como se explicará más adelante.

TinyDB es una herramienta interesante, y posiblemente representa el estado del arte en gestión de la información en redes de sensores inalámbricos, y es por ello que se ha decidido realizar un estudio más en profundidad de esta plataforma.

Además de ello, TinyDB implementa un gran conjunto de optimizaciones para realizar un manejo eficiente la WSN. Se analizará cada una de las optimizaciones realizadas por TinyDB con el objetivo de aprender y tomarlas de ejemplo para otras posibles implementaciones en otras plataformas durante el desarrollo de la parte práctica de este proyecto.

Durante este apartado se explicará el funcionamiento interno de TinyDB así como las optimizaciones implementadas por dicha plataforma.

2.4.1. Modelos de datos en TinyDB

En el modelo de datos de TinyDB se ve la red de sensores como si fuera una única tabla cuyas filas contienen los siguientes datos:

NodeId	TimeStamp	Col1	……	ColN

Tabla 1: estructura filas tabla sensors de TinyDB

El primer campo corresponde al id del nodo, seguido por el segundo campo que es el *timestamp* de la lectura de datos. Posteriormente vienen una serie de columnas. A cada tipo de sensor físico de los nodos (cada atributo) le corresponde una columna.

Respecto a las filas de la tabla, a cada fila le corresponde un nodo y un instante de tiempo concreto, es decir, es la medición de los sensores de un nodo concreto en ese momento concreto. Si un sensor no tiene cierto tipo de atributo, esa columna contendrá NULL para su lectura (puede ser que otro nodo sí complete ese campo por tener ese tipo de sensor).

Las mediciones son sólo adquiridas cuando se necesitan y son almacenadas por un corto espacio de tiempo, por lo que a diferencia de una base de datos SQL, esta tabla no es una base de datos permanente. TinyDB consigue dar a la red de sensores el aspecto de una base de datos sobre la que se puede lanzar *queries*, pero los datos son volátiles, están ahí durante un corto espacio de tiempo en torno al momento en que son capturados.

¿Dónde quedan almacenados estos datos temporalmente? En los llamados puntos de almacenamiento, que están en cada nodo y son persistentes solamente hasta que se llenan, momento en que empiezan a sobrescribirse datos antiguos. Se hablará de ellos más adelante.
[9]

Para poder crear un modelo de datos para una WSN concreta, los nodos envían el llamado catálogo de metadatos al centro de control periódicamente. Dicho catálogo es usado para optimizaciones, modelado de datos, etc… La información que contiene dicho catálogo es la siguiente:

Atributos locales	• Coste de muestreo • Tiempo

	• Energía
Eventos	• Nombre • Signature • Frecuencia
Funciones definidas por usuario (predicados)	• Nombre • Signature • Selectividad (estimada)
Sistema extensible de agregados	• Nombre de los agregados • Puntero a sus códigos • Los agregados consisten de una tripleta (initialize, merge, update)
Características	• Monotonic • Exemplary • Summary

Tabla 2: información en el catálogo de metadatos

2.4.2. Tipos de *queries* en TinyDB

En TinyDB se pueden encontrar los siguientes grupos de *queries*:

- ***Queries* de monitorización**: Pregunta el valor de uno o más atributos continuamente o periódicamente.

- ***Queries* de estado de la red**: *Meta-queries* sobre el estado de la propia red.

- ***Queries* de exploración**: *Queries* para examinar el estado de un nodo particular o un conjunto de nodos, en un momento concreto.

- ***Queries* anidadas**: Solamente eventos y puntos de materialización aceptan *queries* anidadas. TinyDB no acepta otras queries anidadas, pero es posible emular sus efectos guardando los datos en puntos de materialización con una query para usarlos más tarde en otra.

- ***Action queries***: Con la orden "ouput action" es posible invocar un comando externo cuando una tupla satisfaga la query que lleva el comando. Por ejemplo select…from…where…output action…sample period… Se verá más adelante.

- ***Offline deliveries***: Cuando los resultados son capturados más rápidos de lo que pueden ser transmitidos al centro de control, TinyDB soporta el almacenamiento de resultados en una EEPROM para envío en diferido en vez de en tiempo real.

[10]

2.4.3. Sintaxis de las *queries* en TinyDB

La sintaxis general de las queries de TinyDB es la siguiente:

[ON [ALIGNED] EVENT *event-type*[{paramlist}] [*boolop event-type*{paramlist} ...]] SELECT [NO INTERLEAVE] <expr>| *agg*(<expr>) | *temporal agg*(<expr>), ... FROM [sensors	*storage-point*], ... [WHERE {<pred>}] [GROUP BY {<expr>}] [HAVING {<pred>}] [OUTPUT ACTION [*command* | SIGNAL *event*({paramlist}) | (SELECT ...)] | [INTO STORAGE POINT *bufname*]] [SAMPLE PERIOD *seconds* [[FOR *nrounds*] | [STOP ON *event-type* [WHERE <pred>]]] [COMBINE { *agg*(<expr>)}] [INTERPOLATE LINEAR]] | [ONCE] | [LIFETIME *seconds* [MIN SAMPLE RATE *seconds*]] CREATE [CIRCULAR] STORAGE POINT *name* SIZE [*ntuples*	*nseconds*] [(*fieldname type* [, ... , *fieldname type*])] | [AS SELECT ...] [SAMPLE PERIOD *nseconds*]

A continuación ejemplos de las *queries* que se pueden realizar en TinyDB:

Query para eliminación de puntos de almacenamiento en los nodos	DROP STORAGE POINT *name*

Query para selección de valores capturados por los sensores ante la ocurrencia de un evento y especificando un período de muestreo	SELECT s.a1 FROM sensors AS s, events AS e WHERE s.nodeid = e.nodeid AND e.type = e AND s.time - e.time <= k AND s.time > e.time SAMPLE PERIOD d
Query para selección de valores capturados por los sensores especificando condiciones y período de muestreo	SELECT COUNT(*) FROM sensors AS s, recentLight AS rl WHERE rl.nodeid = s.nodeid AND s.light < rl.light SAMPLE PERIOD 10s
Query para agrupación de valores muestreados por los sensores y cálculo de media de dichos valores, con condición de envío especificada y período de muestreo	SELECT AVG(volume),room FROM sensors WHERE floor = 6 GROUP BY room HAVING AVG(volume) > threshold SAMPLE PERIOD 30s
Query para selección de valores máximos dado un intervalo de captura, con condición de envío especificada y período de muestreo	SELECT WINMAX(light,8s,8s) FROM sensors WHERE mag > x SAMPLE PERIOD 1s
Query para cálculo de media de valores muestreados en un intervalo de tiempo dado	SELECT WINAVG(volume, 30s, 5s) FROM sensors SAMPLE PERIOD 1s
Query para muestreo de sensores especificando un lifetime (para adaptar el período de muestreo a un valor que asegure un tiempo de vida del nodo de los días especificados)	SELECT nodeid, accel FROM sensors LIFETIME 30 days
Query para muestreo de sensores especificando un lifetime	SELECT a1, ... , anumSensors FROM sensors WHERE p LIFETIME l hours

Query para muestreo de sensores especificando una condición para lanzar una acción a realizar cuando se cumpla dicha condición	SELECT nodeid,temp WHERE temp > thresh OUTPUT ACTION SIGNAL hot(nodeid,temp) SAMPLE PERIOD 10s
Query para muestreo de sensores especificando una condición para lanzar una acción a realizar cuando se cumpla dicha condición	SELECT nodeid,temp FROM sensors WHERE temp > threshold OUTPUT ACTION power-on(nodeid) SAMPLE PERIOD 10s
Query para muestro de sensores ante la ocurrencia de un evento especificado	ON EVENT bird-detect(loc): SELECT AVG(light), AVG(temp), event.loc FROM sensors AS s WHERE dist(s.loc, event.loc) < 10m SAMPLE PERIOD 2 s FOR 30 s
Query para eliminar un muestreo ante evento especificado previamente	STOP ON EVENT(param) WHERE cond(param)
Query para creación de puntos de almacenamiento en los nodos	CREATE STORAGE POINT recentlight SIZE 8 AS (SELECT nodeid, light FROM sensors SAMPLE PERIOD 10s)
Query para crección de árbol semántico de enrutamiento sobre la red especificada	CREATE SRT loc ON sensors (xloc,yloc) ROOT 0
Query para crección de árbol semántico de enrutamiento sobre la red especificada	CREATE SRT stemp ON sensors(temp) ROOT N5

Tabla 3: sintaxis de las queries de TinyDB

[10]

2.4.4. Bases de la gestión de la red en TinyDB

TinyDB se ejecuta sobre motas MICA (TinyOS) por lo que todos los nodos de la red serán de este tipo. Todas las comunicaciones en una red ejecutando TinyDB son realizadas en *broadcast*.

Aunque las comunicaciones se realizan en *broadcast*, el sistema operativo filtra los mensajes de manera que los mensajes pueden ser dirigidos a un único nodo, pero los nodos vecinos son capaces de ver los mensajes hacia los nodos que no son ellos mismos si están despiertos en el momento de la transmisión y cerca del nodo que transmite. A esta característica se la conoce como "*snoop*" y más adelante se verá que existen una serie de optimizaciones basadas en dicha característica.

En TinyDB existe la confirmación de mensaje recibido a nivel de enlace, es decir, directamente al vecino más cercano que te entrega el mensaje, pero no hay confirmación extremo a extremo.

En cuanto a la topología de red, ésta debe ser descubierta automáticamente por los dispositivos en vez de ser fijada en el momento de implantación de la red. Más adelante se verán los SRT (*Semantic Routing Trees*) y se hablará de la topología en forma de árbol de las redes basadas en TinyDB.

En este tipo de redes los nodos guardan información de la red, por ejemplo una pequeña lista de vecinos escuchados recientemente, información de conectividad de dichos vecinos a la red y calidad del enlace con cada vecino.

Como se decía anteriormente la topología de la red en redes ejecutando TinyDB es en forma de árbol. La raíz del árbol de la red es la estación base, también llamada "*sink*" o centro de control. En algunos casos cabe la posibilidad de que existan múltiples árboles de enrutamiento, en el caso de que uno o varios nodos estén conectados a más de un padre.

La transferencia de datos funciona siempre de esta manera. La estación base envía peticiones de *request* a los hijos, que escuchan dichas peticiones, las procesan y las reenvían a sus hijos, hacia abajo en el árbol.

Tras procesar las peticiones de *request*, si corresponde, los nodos responden con los resultados (lecturas realizadas sobre los sensores, etc…) y estos son reenviados hacia la estación base, hacia arriba en el árbol.

Una parte muy importante en la gestión de red llevada a cabo por TinyDB son los llamados SRT, que se han referenciado previamente en este documento y sobre los que se profundizará

ahora. Los SRT son una serie de tablas que son construidas para conocer de antemano, sin necesidad de propagar *queries* hacia abajo en el árbol de la red, los valores o rangos de valores que cada nodo y sus hijos pueden tener en un atributo concreto. Esto es muy útil a la hora de optimizar el uso de la red, porque de esta manera los padres pueden saber si alguno de sus hijos podría participar o no en una *query* recibida y decidir si transmitir dicha *query* en esa rama del árbol. Cada nodo dispone de intervalos unidimensionales que representan el rango de valores que un atributo concreto puede tener.

Los SRT se construyen en dos fases. Primero, desde la raíz del árbol, se inunda la red con una petición de tipo SRT *build request*, que incluye el nombre del atributo para el que se va a construir dicho SRT. A continuación, cuando los hijos reciben el *request*, estos responden con sus valores o rangos de valores y cada padre de la red va creando su propio rango en base a los valores que él puede transmitir para ese atributo junto con los valores de todos sus hijos, y lo propaga hacia arriba en la red hasta llegar a la estación base.
[8]

2.4.5. Estrategias de optimización de TinyDB

Como se decía al comienzo de este análisis teórico sobre TinyDB una de las motivaciones principales para analizar esta plataforma es entender las diferentes optimizaciones que implementa en cuanto a uso de recursos, tráfico de red y consumo de batería en los dispositivos.

En los próximos puntos se analizará cada optimización, agrupadas en capítulos según el tipo de estrategia utilizada en dicha optimización (estrategias locales de los nodos, estrategias de red, etc…).

2.4.5.1. Estrategias de optimización de TinyDB basadas en el uso de la red

Manejo de metadatos

Como ya se dijo en la introducción de TinyDB en esta plataforma cada nodo envía un conjunto de metadatos que describen los atributos locales, eventos, funciones definidas por el usuario, agregados y otras características del nodo en cuestión.

Esta información es utilizada para optimizar las *queries* mediante la utilización del coste de muestreo de cada atributo y la utilización del rango de los atributos por ejemplo para construir el SRT. También es posible estimar gracias a estos metadatos el tiempo de vida que deben tener las *queries*. Entre los metadatos se encuentra el coste de procesar los datos y el coste de enviar los datos, por lo que gracias a esta estimación de costes se puede saber aproximadamente cuanto tiempo se necesita dejar una *query* activa.
[10]

Snooping

Como ya se dijo, en TinyDB las comunicaciones se realizan en *broadcast*, y aunque un mensaje vaya dirigido a un nodo en cuestión, los vecinos (principalmente los más cercanos) pueden escuchar las peticiones y respuestas de dicho nodo. *Snooping* consiste en esto, en escuchar las respuestas de los nodos vecinos para realizar varios tipos de optimizaciones que se describen a continuación.

Mediante la técnica de *snooping* es posible evaluar predicados de tipo agregado (por ejemplo la *query* AVG para calcular valores medios, la *query* WINMAX, etc…) para suprimir localmente valores muestreados de los sensores si el nodo sabe que no va a poder contribuir a la *query* porque un valor de un vecino anularía su propio valor. Por ejemplo, con la orden MAX en una *query* sobre un atributo A, si el nodo N escucha a un vecino enviar un valor de A mayor que su propia lectura, automáticamente descartará su propio valor o asignara una prioridad muy baja a su entrega (para entregar otras cosas primero), porque no puede contribuir a la *query* MAX (el valor de su vecino hará inservible su propio valor). De esta manera se reduce el tráfico de red y se optimiza el consumo de energía, ahorrando en el envío de datos innecesarios.

Otro ejemplo puede ser el agregado AVERAGE. Los nodos podrían asignar prioridades más bajas si escuchan que sus vecinos han enviado valores parecidos a los que ellos mismos podrían enviar, ya que si es así, no van a contribuir demasiado al agregado AVERAGE que calcula la media de los valores enviados.

Son técnicas muy útiles ya que se reduce en tráfico de red y se ahorra en consumo de energía en varias de las motas que no necesitarán enviar hasta la estación base los datos innecesarios.
[10]

Semantic routing trees (SRTs)

Un SRT es una tabla o árbol con información sobre cada nodo. El SRT es diseñado para permitir a cada nodo determinar si alguno de los nodos que cuelgan de él podría participar en una determinada *query*.

Cada nodo almacena para cada atributo un intervalo unidimensional de valores que representan el rango de valores que el atributo puede tener, contando con su propia contribución y la de todos sus hijos.

La construcción del SRT se realiza desde la raíz. La red se inunda hacia abajo con peticiones de tipo SRT *Build Requests*. El comando de TinyDB para realizar dicha tarea es el siguiente: CREATE SRT loc ON sensors(xloc, yloc) ROOT 0.

Los nodos hijos responden al request con sus propios valores y los de todos sus hijos, y cada padre construye su propio intervalo unidimensional y se lo envía a su vez a su propio padre, y así repetidamente hasta llegar a la raíz de la red.

Se debe realizar un mantenimiento del SRT, ya que desde que es calculado cuando se inunda la red de peticiones, podría haber habido cambios en la red (por ejemplo una mota terminal conecta a otro nodo y por lo tanto cambia de padre en el árbol).

Si un nodo cambia de padre, dicho nodo envía a su nuevo padre un mensaje de selección de padre. Los nodos notifican al nuevo padre para actualizar su rango unidimensional y que éste lo propague hacia la raíz. El padre que ha perdido el hijo no sabrá durante un tiempo que lo ha perdido, pero pasado ese tiempo asumirá que ha perdido a dicho nodo hijo y borrará su entrada del SRT.

Hay un coste de construcción y mantenimiento asociado a los SRT. Normalmente tener un SRT conviene, pero en algunos casos podría ser mejor no incurrir en el coste de construirlo y mantenerlo.

Se va a ver un ejemplo de utilización de SRT. Se realiza la *query* SELECT Light where x>3 and x<7.

Abajo se puede ver un gráfico de programación de la *query* hacia abajo en la red. En este caso el nodo 1 no propagará la query al nodo 2 porque sabe que el nodo 2 no puede contribuir a esta query porque X no está en el rango del WHERE. Además el nodo 3 no enviará la *query* al nodo 5 por la misma razón.
[10]

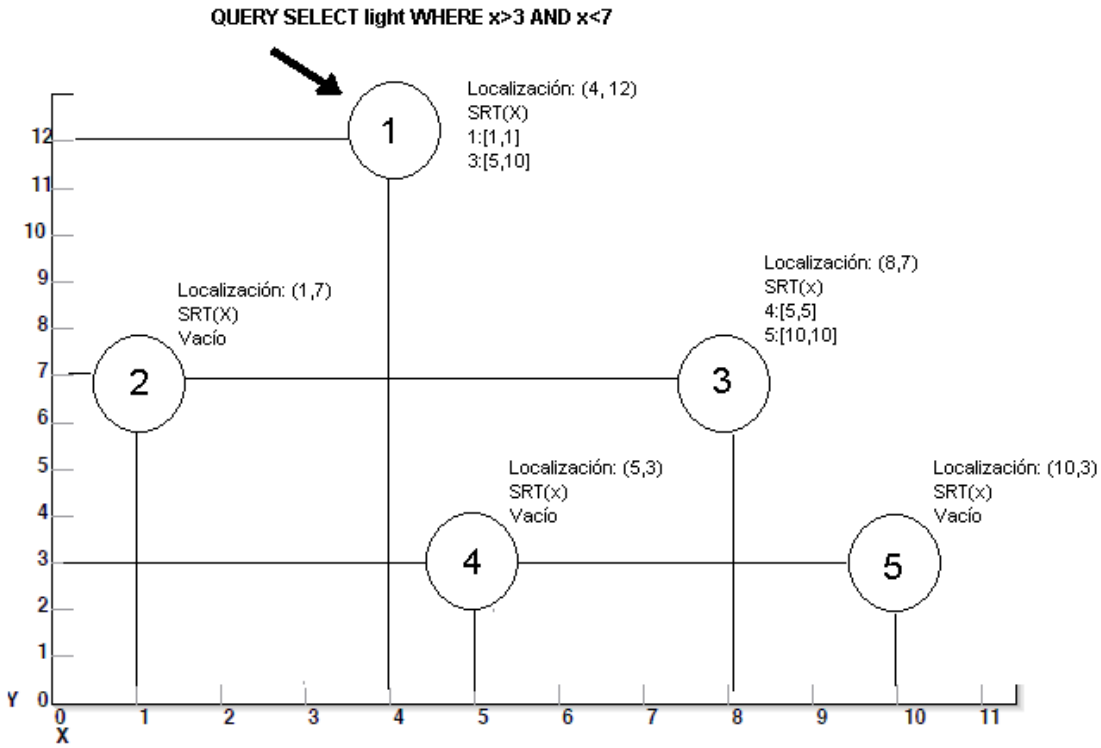

Figura 10: ejemplo de utilización de árboles semánticos en TinyDB

2.4.5.2. Estrategias de optimización de TinyDB locales en cada nodo

Política de ejecución de *queries*, EPOCHs

Una característica fundamental de las WSN es que los nodos duermen la mayor parte del tiempo, y despiertan para realizar medidas de sus sensores cada cierto tiempo. El período de tiempo entre el comienzo de cada muestro se conoce como EPOCH (*once per time-period*). Las tuplas son producidas en intervalos de EPOCH.

Los nodos en TinyDB ejecutan un protocolo simple de sincronización para ponerse de acuerdo sobre una base de tiempo global que les permita empezar y terminar cada EPOCH a la misma vez.

Los nodos duermen la mayor parte de un EPOCH y se despiertan para realizar el muestreo. Al comienzo del EPOCH despiertan, a continuación realizan el muestreo de los sensores. Seguidamente aplican los operadores a los datos generados localmente y recibidos por sus vecinos. Por último devuelven los resultados al padre, que, debido a que los nodos están

sincronizados, los padres estarán despiertos también para recibir cuando los hijos transmitan. El tiempo *t_awake* que están despiertos debe ser lo suficientemente grande para asegurar el funcionamiento correcto, y dependerá del número de nodos transmitiendo.

Como ya se ha dicho un EPOCH es un intervalo mediante el cual todos los nodos de la red se sincronizan para realizar los muestreos, los envíos de datos, etc… Pero también se realiza una subdivisión de EPOCH.

Existe una problemática, principalmente con *queries* de tipo *aggregate*, y es que se necesita que los padres tengan todas las lecturas de los nodos hijos antes de realizar el cálculo del agregado. Para este propósito los EPOCH se subdividen en intervalos más pequeños y a cada nodo se les asigna un intervalo dentro del EPOCH basado en su posición en el árbol de enrutamiento. A esto se le conoce como "*slotted approach*".

Los intervalos se asignan en orden inverso de manera que 1 es el último intervalo en el EPOCH. Cada nodo es asignado al intervalo igual a su nivel en el árbol (número de saltos hasta la raíz).

Durante el intervalo de un nodo, si está ejecutando un *aggregate*, el nodo computa el cálculo parcial consistente de la combinación de los valores de sus hijos que el nodo ha escuchado, con sus propias lecturas. Luego el nodo transmite hacia arriba su cálculo parcial o directamente las lecturas suyas y de sus hijos.

A continuación se puede ver gráficamente como los nodos transmiten en su intervalo permitiendo una mejor sincronización entre los nodos más alejados de la raíz y los más cercanos.
[10]

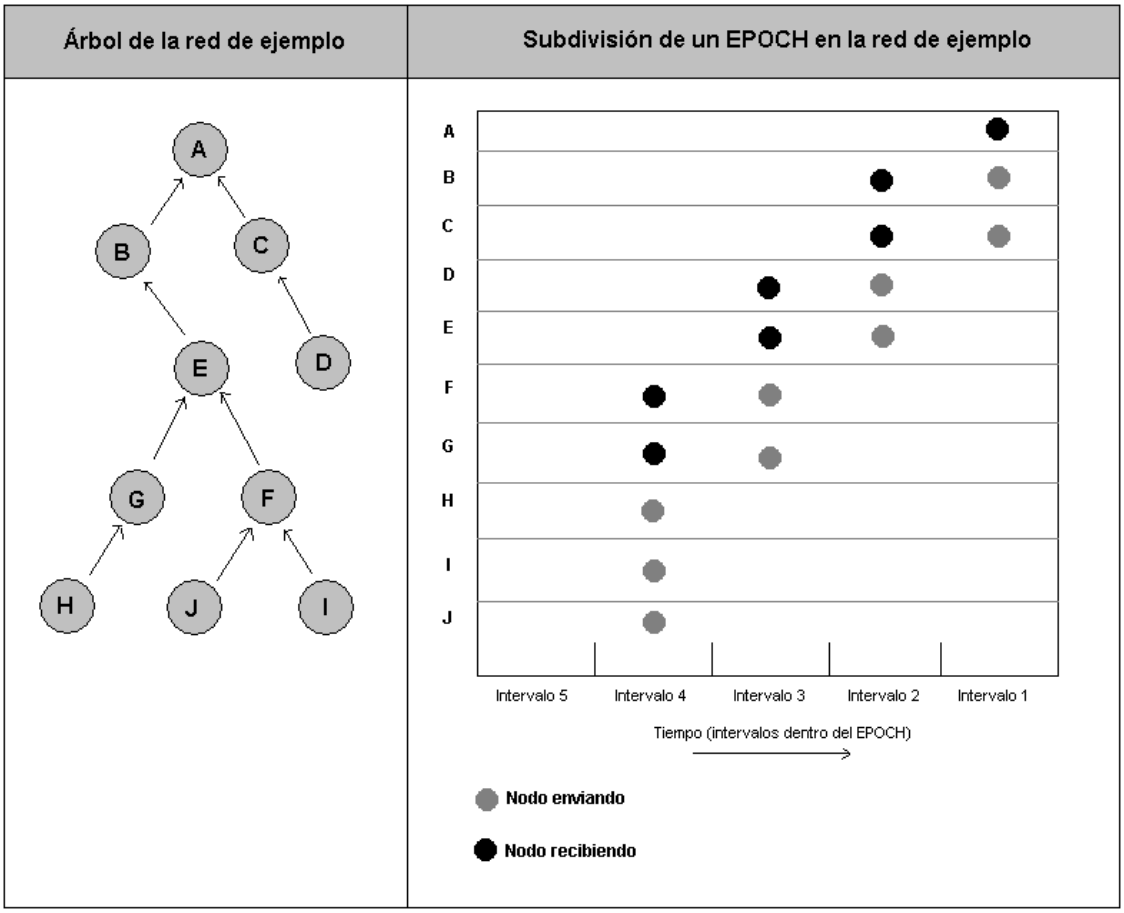

Figura 11: ejemplo de utilización de EPOCHs en TinyDB

Política de ejecución de *queries*: *queries* múltiples

En TinyDB, cuando hay múltiples *queries* en curso, se comparten los SRTs entre las *queries*, pero el muestreo de sensores no es compartido (si dos *queries* necesitan una lectura de sensores con unos pocos milisegundos de diferencia entre ellas, esto causará dos lecturas de sensores aunque haya poco tiempo entre cada lectura). La transmisión de datos tampoco es compartida (para cada *query* se transmiten los datos independientemente).

Esto podría ser un punto a estudiar para futuras optimizaciones de la plataforma.
[10]

Política de ejecución de *queries*: priorización de entrega de datos

La problemática es la siguiente. Después de haberse muestreado los sensores y haber sido aplicados los operadores implicados en las *queries*, los resultados son encolados en la cola de

radio, para ser entregados al nodo padre. En algunos casos esta cola se llena y hay que tomar decisiones sobre los envíos.

La solución que toma TinyDB es enviar los resultados que optimizan la calidad de la respuesta que llega al usuario (al nodo raíz). Existen varios esquemas para decidir cómo priorizar la entrega de datos o como resumir dicha información para enviar el máximo posible.

En *queries* de selección se pueden usar los siguientes esquemas:

- **Esquema *naive***. En este esquema ninguna tupla es considerada más importante que las otras. Las *queries* se encolan en FIFO y las queries se descartan si la cola está llena.
- **Esquema *winavg***. Es similar al esquema *naive*, pero en vez de eliminar resultados cuando la cola se llena, los dos resultados de la cabeza de la cola son promediados y convertidos en un único resultado que es la media de ambos, dejando así hueco para el nuevo resultado.
- **Esquema *delta***. En este caso se asigna a cada tupla una puntuación inicial relativa a su diferencia respecto a la tupla más recientemente transmitida exitosamente desde este nodo, y en cada punto de tiempo, se transmite la tupla con puntuación más alta. Las tuplas con puntuaciones más bajas se transmiten posteriormente cuando la cola ya no está llena. Pueden transmitirse desordenadamente. Este esquema responde al pensamiento intuitivo de que grandes cambios seguramente darán información más interesante. Por esa razón se transmiten prioritariamente tuplas con valores muy diferentes a la última transmitida.

En *queries* de tipo *aggregate*:

Snooping. Ya se comentó antes sobre esta técnica. Se escucha lo que transmiten los nodos vecinos, y localmente se suprimen valores que no pueden aportar nada a los *aggregates* en curso.
[10]

Lifetime – adaptación de ratios para adaptar consumo de energía

Como ya se vio en la sintaxis del lenguaje, en las *queries* de TinyDB es posible especificar el período de tiempo con el que quiere que se esté realizando el muestreo de sensores especificado por la *query*.

El modificador *lifetime* sirve para especificar el período de otra manera diferente. Lo que pedimos al nodo con *lifetime* es que realice el muestreo de manera que garantice un tiempo de vida (con batería) mínimo para el nodo. Por ejemplo: Select nodeid, accel from sensors

lifetime 30 days. Con ello se está diciendo al nodo que muestree al máximo posible, pero garantizando un tiempo de vida de 30 días.

Se elige por tanto un ratio de muestreo y de transmisión apropiados para satisfacer el *lifetime*.

Hay que tener en cuenta que cuando se optimiza una *query*, el optimizador tiene en mente las condiciones de carga de la red, los ratios de muestreos y el *lifetime* solicitados. Pero también toma decisiones estáticas al comienzo de la *query*, las cuales podrían no ser válidas después de días ejecutándose dicha *query*. Por ello es posible que los nodos varíen más adelante los ratios de muestreo y transmisión según sus medidas de energía. El sistema compara la energía restante actual frente a la predicha en el momento de la optimización de la *query*, asumiendo una caída lineal de la energía, y realiza una reestimación del consumo de energía del dispositivo y recalcular los parámetros del *lifetime*.
[10]

Capítulo 3 Especificación del problema a resolver

En los capítulos anteriores se ha hecho una introducción comentando los objetivos de la parte práctica.

Hasta el momento, por una parte se ha hecho una descripción teórica de las redes de sensores inalámbricos: protocolos, tipos de dispositivos, plataformas, etc… Además de realizar un estudio del estado del arte en lo que respecta a gestión de la información en este tipo de redes.

En el capítulo 2 de este proyecto, desde los apartados 2.1 a 2.3, ambos incluidos, se ha hecho la descripción teórica. Durante todo el punto 2.4, se ha realizado un análisis de TinyDB, por ser la plataforma que se ha considerado más avanzada en cuanto a gestión de la información en redes de sensores inalámbricos, y es la que se ha considerado mejor representa el estado del arte actual en este campo.

Por otra parte, se ha marcado como objetivo realizar una aplicación para redes de sensores inalámbricos basada en la plataforma Arduino, utilizando la plataforma Raspberry Pi como centro de control.

En este apartado se describirá el problema que dicha aplicación pretende resolver y los objetivos que se pretenden alcanzar.

3.1. Aplicación para monitorización de consumo energético

Se pretende realizar un sistema cuyo objetivo es monitorizar el consumo energético de diferentes subzonas de una zona monitorizada.

Para hacer esto posible el sistema debe constar de elementos medidores, que serán capaces de tomar muestras del consumo de la subzona que monitorizan, para luego enviar dicha medición, junto con el identificador de la subzona, al centro de control.

En lo que compete a este proyecto, el centro de control solamente será un elemento centralizador, raíz de la red formada por los elementos medidores, que recibirá y organizará en una base de datos la información recibida por dichos medidores, y ofrecerá una interfaz al usuario.

Sin embargo es interesante que la implementación del centro de control se realice sobre alguna plataforma más potente y ampliable, sobre la que, en un futuro, se pudiera desarrollar otro tipo de funcionalidades, como por ejemplo la capacidad de enviar las medidas de consumo recopiladas a un servidor externo a través de Internet.

A continuación se puede ver un diseño simplificado de lo que podría ser a grandes rasgos la aplicación a desarrollar.

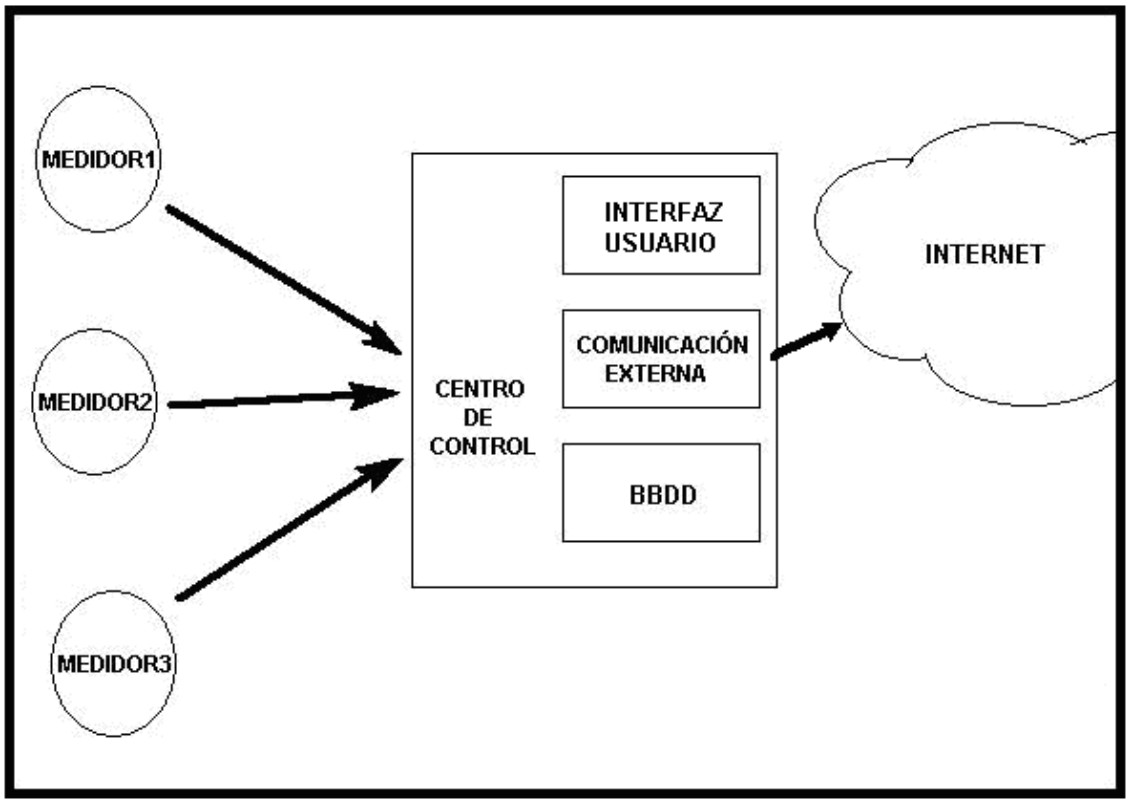

Figura 12: esquema del sistema de monitorización de consumo

Para el desarrollo de este sistema será suficiente con disponer de una topología de red en forma de estrella, donde los nodos medidores solamente se comunicarán con el centro de control y no será necesario que se comuniquen o coordinen entre ellos ni que implementen complicados protocolos de enrutamiento.

Otra característica deseable es que los sensores utilizados para la medición de consumo sean no invasivos, para facilitar la implantación y movilidad del sistema.

Cada nodo medidor debe tener un valor fijo que le representa, un identificador que irá fijado en el propio código fuente del medidor. En el momento de implantación del sistema debe asegurarse que cada nodo tenga su identificador propio y único.

Otro requisito de esta aplicación es que sea capaz de funcionar en distintos entornos, con distintos niveles de consumo, voltajes y potencias contratadas, etc… Por ello, a la hora de elegir el hardware a utilizar, se debe tener en cuenta que la aplicación podría monitorizar entornos industriales, entornos domésticos, etc… y que dicho hardware debe ser flexible en su capacidad de tomar mediciones. Sin embargo será suficiente con que los medidores sean capaces de medir corriente alterna, ya que el objetivo de medición será el suministro eléctrico.

Por último, es deseable que las mediciones guardadas en la base de datos lleven una marca de tiempo y de subzona, de manera que quede perfectamente identificada la medición en tiempo y espacio.

3.2. Caso de uso centro de control

Aunque en un futuro pudiera realizarse una versión mejorada del centro de control, en la aplicación a desarrollar se dispondrá de un centro de control simplificado, que tendrá implementada toda la funcionalidad necesaria para comunicarse y recopilar muestras de la red de sensores, pero que sólo ofrecerá una sencilla interfaz local hacia el usuario, mediante la cual el usuario podrá consultar las muestras recibidas en tiempo real, así como salvarlas hacia o recuperarlas desde una base de datos.

Por ello, este centro de control implementará una sencilla base de datos sobre un fichero, en la que podrá almacenar las muestras recibidas o recuperar las muestras guardadas previamente.

A continuación se analiza el caso de uso del centro de control y las acciones que el usuario puede realizar sobre el mismo:

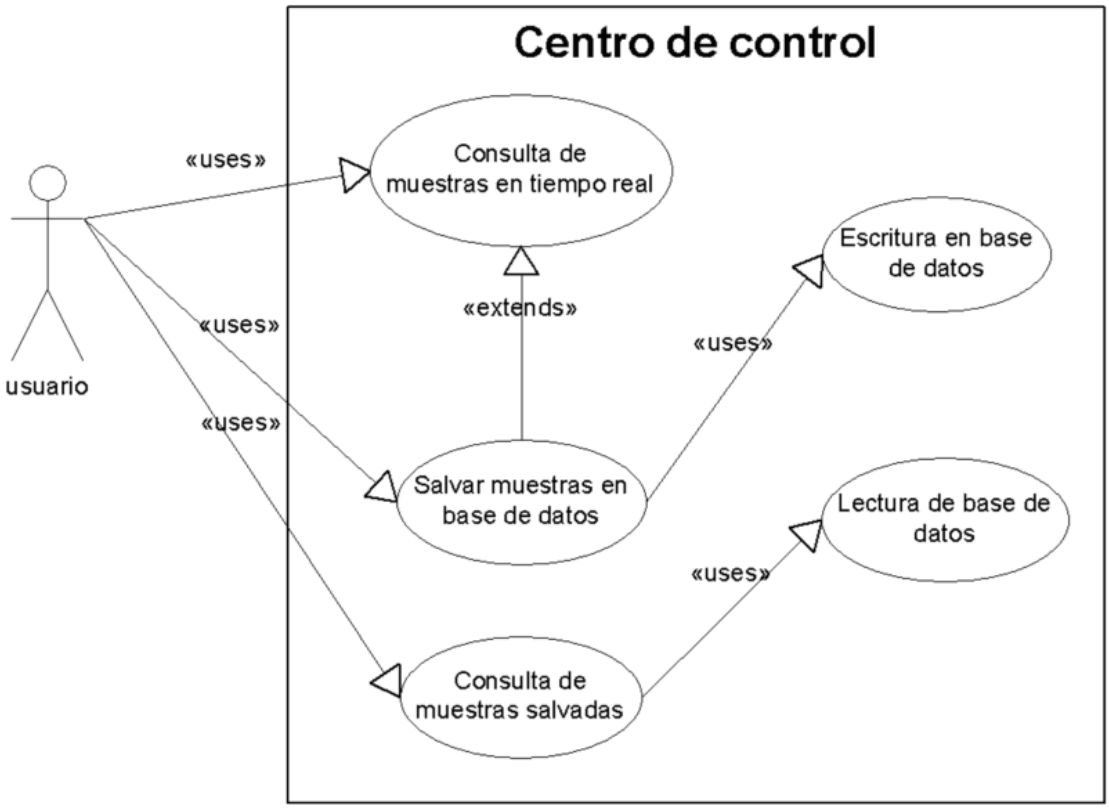

Figura 13: caso de uso centro de control

3.3. Caso de uso red de sensores

La red de sensores vista como un sistema propio debe ofrecer una interfaz de uso hacia el centro de control, de manera que éste pueda consultar las muestras tomadas por el elemento medidor así como, al menos, saber el estado energético del elemento.

A continuación se puede ver el caso de uso de la red de sensores y las acciones que el centro de control puede realizar sobre la misma.

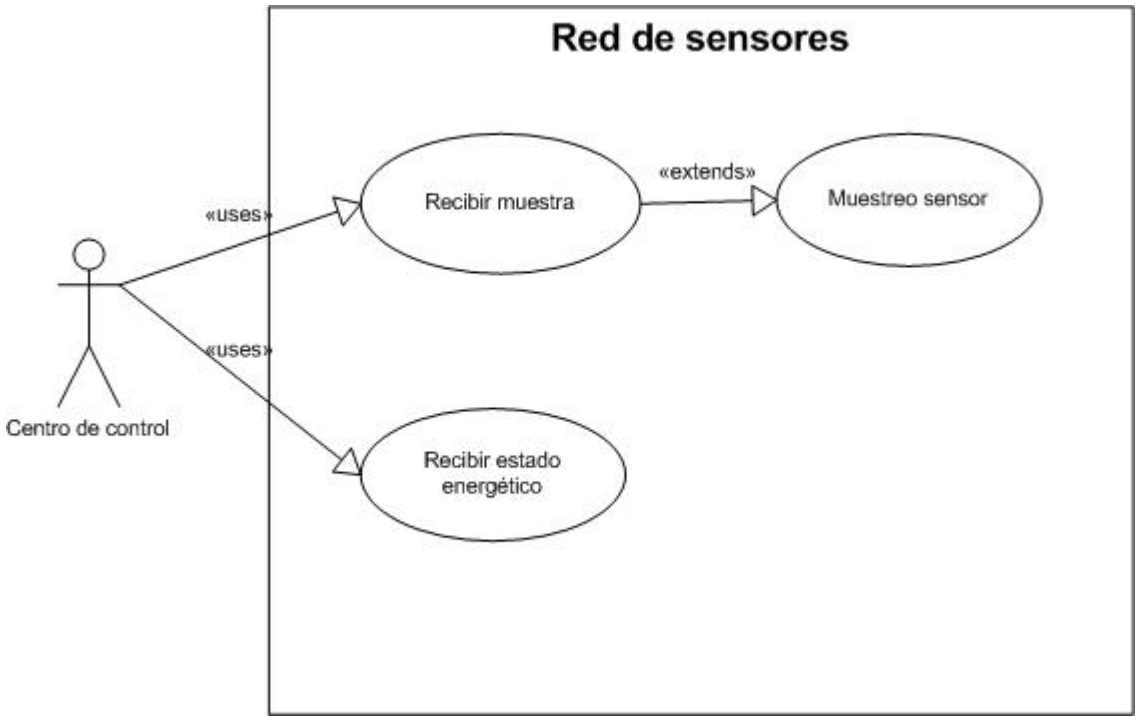

Figura 14: caso de uso elemento medidor

Para el tipo de aplicación que se va a desarrollar es suficiente con una topología de red en forma de estrella, por lo que no será necesario que los elementos medidores se coordinen entre ellos u ofrezcan otra interfaz diferente para la coordinación de la red, sólo necesitarán comunicarse con el centro de control.

3.4. Márgenes de error y optimizaciones

Al comienzo de esta sección se comentó que una de las características del sistema es que los sensores para realizar las mediciones de consumo energético fueran no invasivos. Este tipo de sensores, pese a realizar mediciones bastante exactas, suelen funcionar con un cierto margen de error.

Puesto que se pretende medir el consumo medio en las zonas monitorizadas, no es necesaria una alta precisión en los valores tomados, siendo preferible admitir un pequeño margen de error a favor de la facilidad y comodidad de utilizar sensores de corriente no invasivos.

Por otra parte se intentará que la aplicación funcione de la manera más óptima en la medida de lo posible, evitando generar tráfico de red innecesario, intentando disminuir el consumo de los dispositivos medidores con el fin de alargar la vida de los mismos, etc…

Capítulo 4 Diseño de la aplicación

4.1. Elección del hardware a utilizar

Antes de comenzar con el desarrollo de la aplicación para monitorización de consumo energético, se debe realizar un estudio del hardware más apropiado para ser usado en el sistema.

Se debe tener en cuenta varios factores para tomar una decisión sobre el hardware a utilizar, por ejemplo:

- El alcance que se necesita para la red de sensores.
- El consumo y duración de los dispositivos medidores.
- La capacidad de recuperación de errores y problemas.
- La facilidad de implementación.
- Las facilidades que debe proporcionar el centro de control.
- Los sensores disponibles para los dispositivos medidores (se necesita un hardware compatible con sensores de corriente eléctrica).
- Los dispositivos elegidos para los medidores deben ser compatibles o debe existir alguna extensión para los mismos que les permita comunicaciones ZigBee a corta distancia.
- El precio.
- Las herramientas, entornos de programación, y facilidades existentes para el desarrollo en la plataforma elegida.

4.1.1. Elección del hardware para los nodos de la WSN

Se ha decidido utilizar la plataforma de hardware libre Arduino por diversas razones:

- Extensible mediante todo tipo de sensores, actuadores, *kits*, placas, etc...
- Precio razonable.
- Soporte para sensores de corriente eléctrica.
- Soporte para extensión ZigBee.
- Bajo consumo energético.
- API y entornos de programación muy completos.

Arduino es una placa a menudo utilizada en ambientes educativos para aprendizaje y desarrollo de proyectos de electrónica. Está basada en procesadores Atmel y desde 2012 también ARM y AVR. Dispone de entradas analógicas y digitales, y existe una gran variedad de extensiones conectables a la placa base que aumentan su capacidad, facilitando la posibilidad de conectar y manejar desde software diferentes sensores y controladores, etc…
[3]

Arduino ofrece un potente entorno de desarrollo que facilita enormemente su programación, y existe una amplia API de programación que permite manejar fácilmente desde un programa para Arduino cualquiera de las extensiones conectables a la placa base.

Existe una amplísima variedad de extensiones para Arduino en forma de kits, placas conectables, sensores, actuadores, etc… Desde placas Ethernet o Wifi conectables que permiten al dispositivo Arduino conectarse a una red local, hasta LCD screens, placas GSM, placas controladoras de motores, sensores acelerómetros o giroscopios, microservos, etc…
[11]

Todas estas características del entorno de desarrollo Arduino dotan a esta plataforma de una gran versatilidad.

El entorno es idóneo para la aplicación que se pretende desarrollar por la facilidad que ofrece para programar este dispositivo en lenguaje Java, así como para controlar fácilmente sensores de corriente eléctrica y placas conectables de comunicaciones ZigBee.

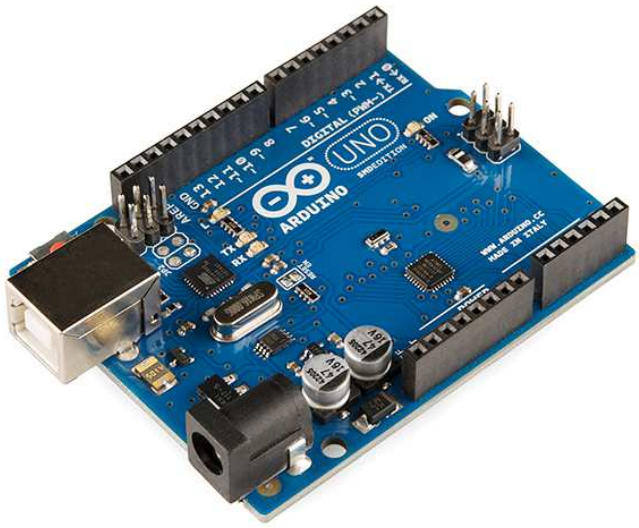

Figura 15: placa base Arduino

4.1.2. Elección del sensor de corriente eléctrica

La elección de un sensor de corriente eléctrica apropiado, es una de las decisiones clave para esta aplicación. Existen una gran variedad de sensores de este tipo. Se debe tener en cuenta varios parámetros a la hora de elegir:

- El rango de intensidades de corriente eléctrica que debe abarcar.
- El margen de ruido que proporciona el sensor elegido.
- La facilidad de programación y de manejo de los datos proporcionados por el sensor.
- La facilidad de calibración del sensor (es imposible fabricar un sensor de este tipo con precisión absoluta por lo que todos estos sensores deben ser calibrados antes de su uso).
- La facilidad de conexión al elemento que se quiere monitorizar (es preferible elegir un sensor no invasivo, que sea suficiente colocar a modo de abrazadera en torno al cable que se pretende monitorizar).

Tras estudiar varias posibilidades se ha optado por el modelo de sensor SCT-013.

Figura 16: sensor SCT-013

Dicho sensor de corriente admite un rango de corrientes desde 50 mA hasta 100A, por lo que es más que suficiente para cualquier de los entornos hacia los que está orientada la aplicación que se pretende desarrollar, ofreciendo además una alta precisión en las mediciones tomadas.

Por otro lado su utilización es sencilla utilizando la placa conectable para Arduino Emontx, de la que se hablará a continuación.

Emontx es una placa que extiende la funcionalidad de la placa base Arduino permitiendo conectar de manera sencilla sensores compatibles.

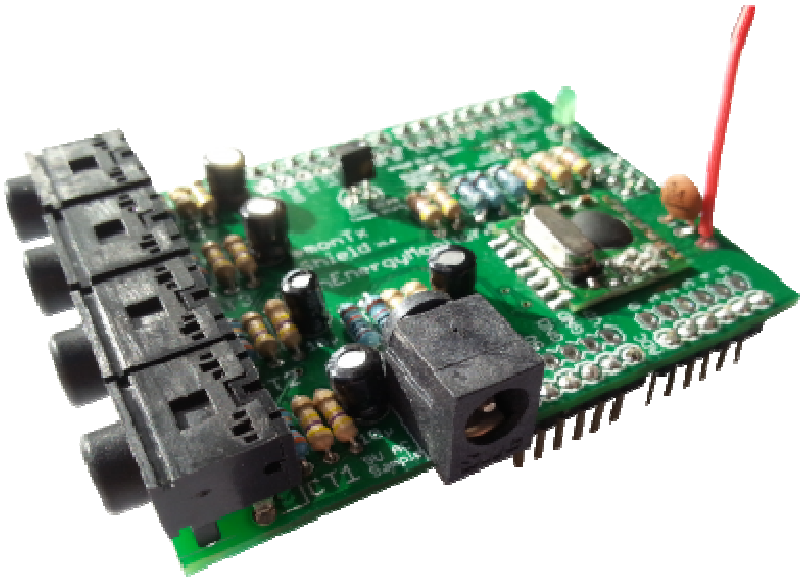

Figura 17: placa Emontx

La placa Emontx se conecta sobre la placa base Arduino. Emontx facilita una interfaz de programación mediante la cual se puede solicitar a la placa Emontx, a la cual debe estar conectado el sensor SCT-013 mediante alguno de sus cuatro conectores, que nos devuelva la medición de corriente tomada.

Otra de las ventajas de la placa Emontx que la hace idónea para la aplicación que se pretende realizar es que es perfectamente compatible con el módulo de comunicaciones ZigBee elegido que se verá a continuación.

4.1.3. Elección del hardware para comunicaciones ZigBee

Para la elección del módulo de comunicaciones ZigBee también se debe tener en cuenta algunos factores:

- El alcance del módulo ZigBee elegido.
- La facilidad de conexión y programación de dicho módulo.

Para dotar a nuestro dispositivo Arduino de la capacidad de comunicarse mediante ZigBee se han elegido dos módulos conectables para Arduino, el Arduino Xbee Shield Pro y el módulo Xbee Series 2.

El módulo Xbee Series 2 es el módulo principal para comunicaciones ZigBee. Se ha elegido este módulo por tener un alcance suficiente para la aplicación que se va a realizar (30 metros en interior y 90 metros en exterior en visión directa).
[14]

En cuanto a la programación, este módulo Xbee para Arduino nos ofrece una gran facilidad de configuración y programación por las siguientes razones:

Además el módulo tiene su propio procesador para gestionar la pila de protocolos ZigBee. Como se verá más adelante es posible configurar una serie de parámetros de red desde un ordenador directamente a cada módulo Xbee, y éste gestionará por sí mismo todo lo relacionado con la red: conexión directa punto a punto con otro dispositivo o conexión con el nodo raíz funcionando en *broadcast*, topología de red en forma de malla o *broadcast*, etc…
[14]
El módulo Xbee conecta a la placa base Arduino mediante el puerto serie, por lo que desde el punto de vista de la programación el envío y recepción de datos se realiza como cualquier otro envío y recepción de datos a través del puerto serie.
[14]

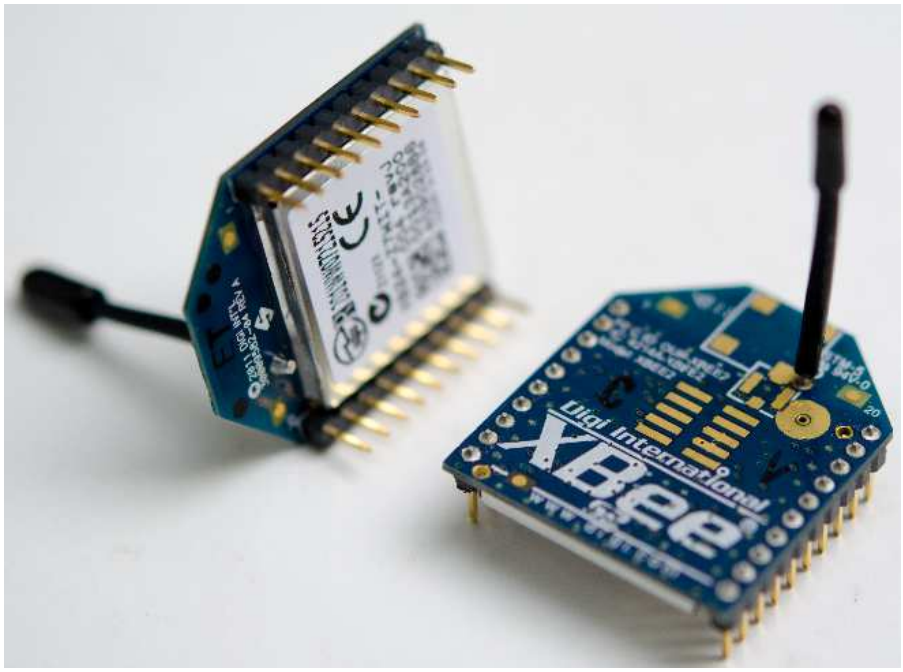

Figura 18: módulo Xbee series 2

La facilidad de conexión la ofrece la placa Arduino Xbee Shield Pro.

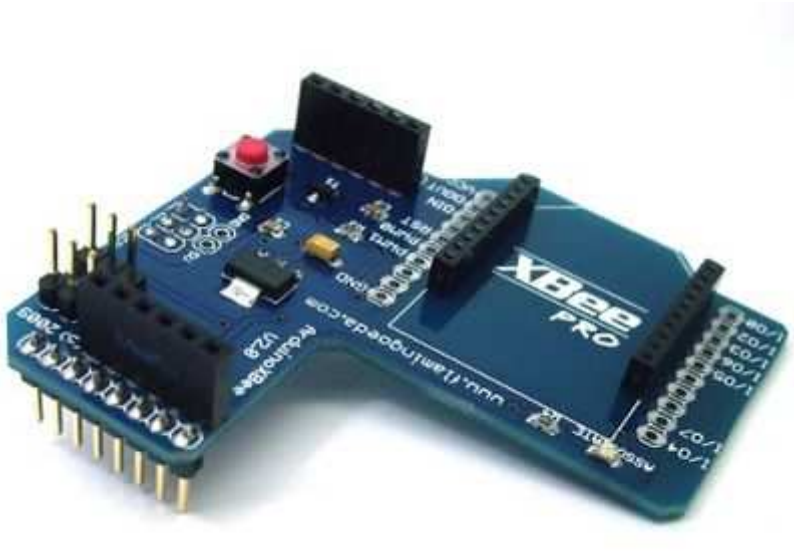

Figura 19: Xbee Shield Pro

Dicha placa Xbee Shield Pro hace de interfaz entre el módulo Xbee y la placa base Arduino, facilitando la conexión a la misma. Dicha conexión se realiza simplemente conectando el Xbee Shield Pro sobre la placa base Arduino y el módulo Xbee directamente sobre el Xbee Shield Pro.
[12]

Figura 20: Conexión módulo Xbee series 2 con Xbee Shield Pro

Además de facilitar la conexión con el módulo Xbee, el Shield reproduce los pines de conexión de Arduino para poder pinchar otras extensiones encima (de esta manera se podrá

conectar nuestra Emontx sobre el Shield). Por otro lado ofrece algunos *jumpers* que permitirán elegir si el puerto serie de la placa base Arduino funciona de manera normal conectado al puerto USB o si está conectado al módulo Xbee de manera que el puerto serie sirva para envío y recepción de datos por ZigBee.
[13]

4.1.4. Elección del hardware para el centro de control

En lo que respecta al hardware del centro de control, se debe diferenciar dos partes. El centro de control con respecto a la red ZigBee de los medidores, y el centro de control como gestor, almacenador y clasificador de información, interfaz hacia el usuario y envío de datos hacia otros servidores.

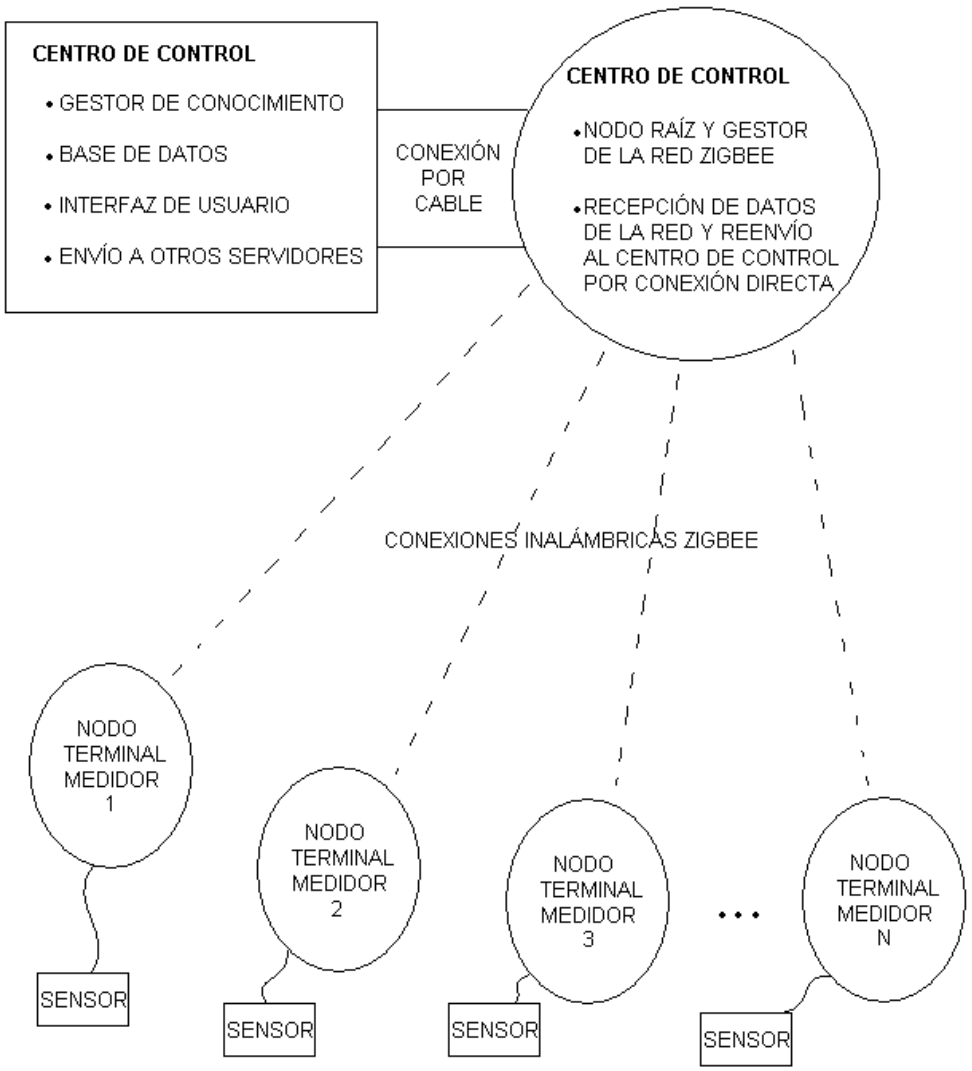

Figura 21: Elementos del sistema y sus conexiones

Teniendo en cuenta esta diferenciación se elegirá la plataforma más adecuada para el centro de control principal, gestor de datos e interfaz de usuario, y el centro de control como nodo raíz de la red ZigBee de terminales medidores.

Para elegir el dispositivo que actuará como nodo raíz de la red ZigBee se debe tener en cuenta algunos factores:

- Soporte ZigBee.
- Conexión lo más estándar posible con otros equipos (para facilitar su conexión al centro de control principal sea cual sea el tipo de equipo o sistema operativo elegido).
- Precio.

Como nodo raíz de la red ZigBee debe elegirse un hardware con soporte ZigBee y compatible con el resto de nodos ZigBee de la red. Existe una gran variedad en el mercado de los llamados *routers* ZigBee, pero su precio suele ser elevado y muchas veces no son tan configurables como interesaría para implementar esta aplicación.

Finalmente se ha optado por utilizar el mismo hardware para el nodo raíz de la red ZigBee que para los nodos terminales, es decir, se utilizará una placa base Arduino programada con un programa diferente (para que actúe como nodo raíz de la red ZigBee) junto con una Xbee Shield Pro y un módulo Xbee Series 2, esta vez configurado con parámetros diferentes a los nodos terminales para que se comporte como coordinador de la red ZigBee. Todo esto es descrito más adelante.

Utilizar una placa base Arduino con módulo ZigBee nos ofrece las siguientes ventajas:

- Totalmente flexible y configurable.
- Permite cargar la placa Arduino con un programa propio que permite un control total de la gestión de la red y la comunicación con el centro de control principal.
- La placa Arduino nos permite conectar el coordinador de la red ZigBee con el centro de control principal mediante una conexión USB totalmente estándar, de manera que en el lado del centro de control principal se puede tener cualquier máquina o sistema operativo.
- El precio es razonable.

Respecto al centro de control principal, se podría utilizar desde un ordenador de sobremesa con sistema operativo Windows, Linux o cualquier otro, hasta un teléfono móvil. Sin embargo interesaría que el centro de control tuviera las siguientes características:

- Preferiblemente de un tamaño pequeño, por lo que se necesitará un sistema empotrado.
- Con algún sistema operativo conocido y completo, en el que se disponga de módulos para implementar cómodamente cualquier funcionalidad que interese, como la interfaz

gráfica para el usuario, la conexión con servidores externos, la gestión de una base de datos, etc…

- Con lenguajes conocidos y entornos de programación que ofrezcan todas las facilidades posibles para facilitar en desarrollo.
- Que tenga un precio razonable.

Entre todas las posibilidades se ha optado por la plataforma Raspberry Pi.

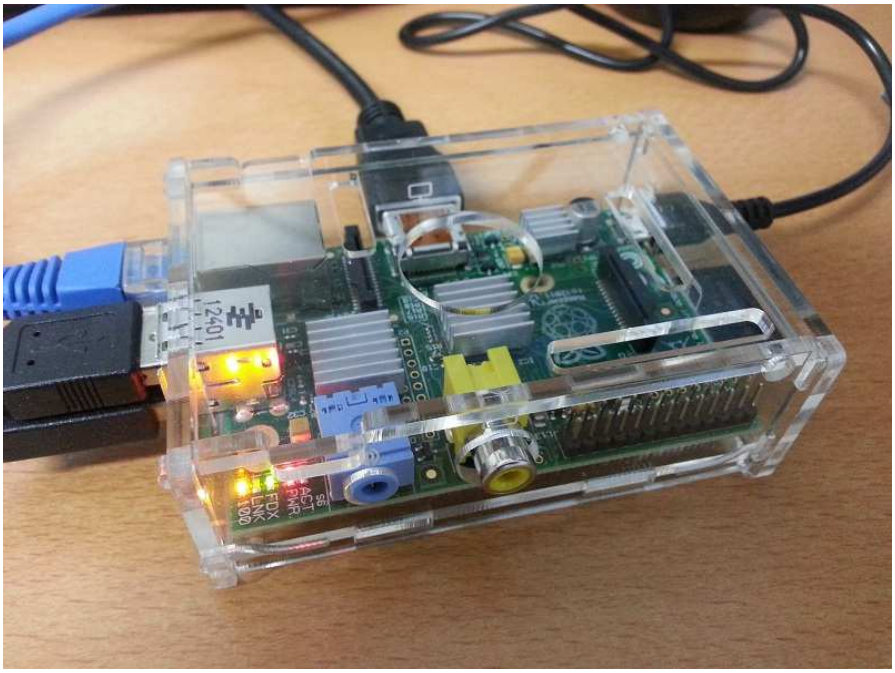

Figura 22: Raspberry Pi

Raspberry Pi es un sistema empotrado basado en procesadores ARM, 512 mb de memoria RAM y un procesador gráfico VideoCore. No incluye disco duro, pues utiliza una tarjeta SD como almacenamiento permanente. La fuente de alimentación también es externa. Como salidas posee dos puertos USB y un controlador Ethernet 10/100. Posee además salida de video RCA y HDMI.

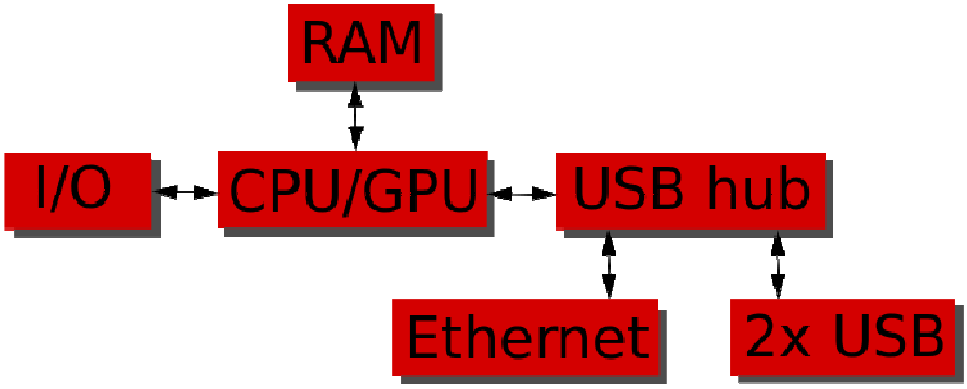

Figura 23: Raspberry Pi diagrama de bloques

La distribución Raspbian para arquitecturas ARM, una versión derivada de Debian, es la más conocida y utilizada como sistema operativo para Raspberry Pi.

Una limitación de Raspberry Pi es que no dispone de reloj en tiempo real, sin embargo puede sincronizar su reloj por red.
[4]

Con estas características, Raspberry Pi permite sin problemas una comunicación directa por USB con el Arduino que actúa como nodo coordinador de la red ZigBee, así como facilita la construcción de una interfaz gráfica para el usuario (ofrecida por ejemplo directamente en un monitor conectado, por red mediante un servidor http, etc…), o la comunicación con un servidor externo a través de Internet.

La configuración de hardware definitiva para nuestro sistema sería entonces:

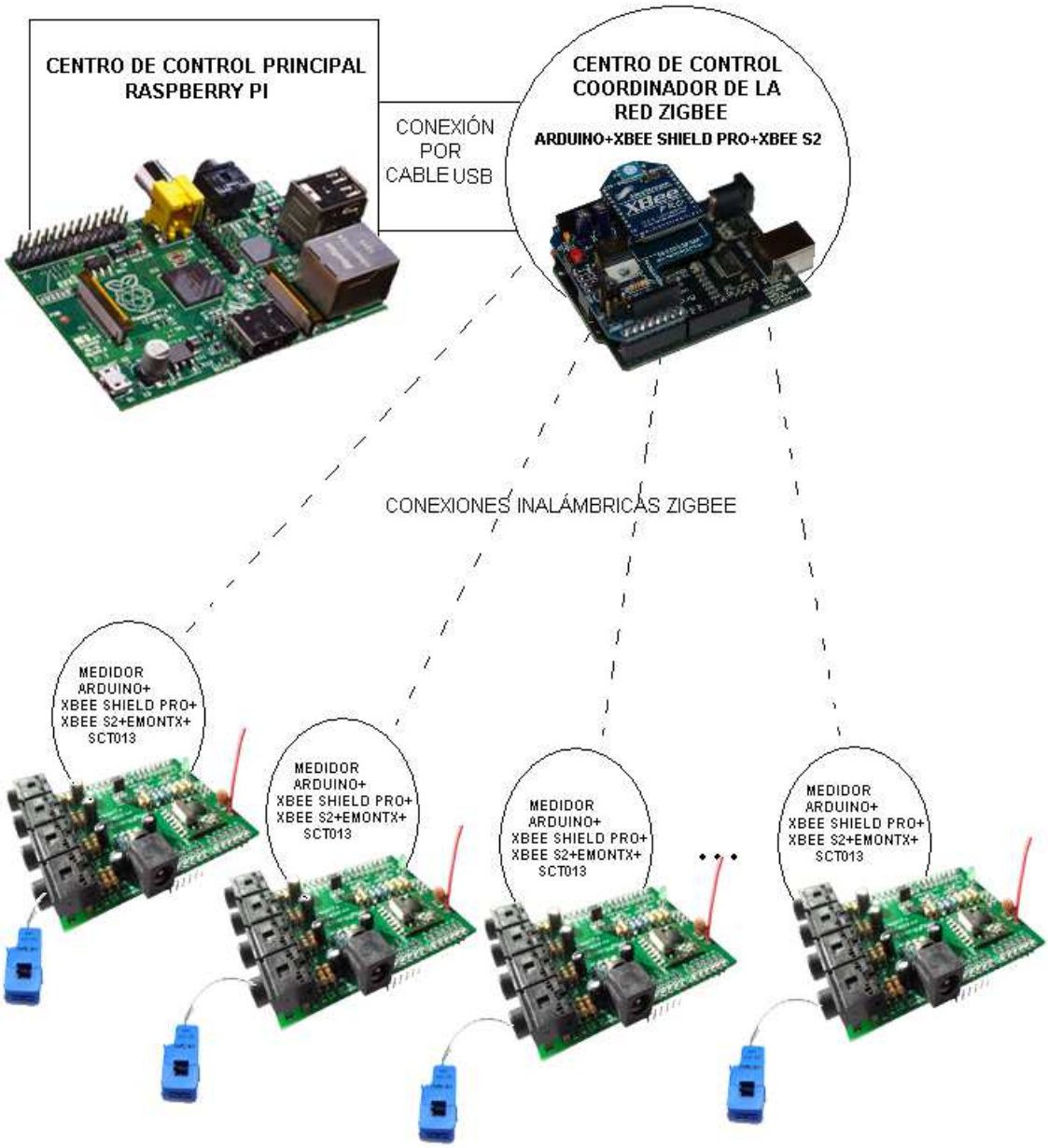

Figura 24: Configuración hardware del sistema

4.2. Elección del entorno de desarrollo a utilizar

En apartados anteriores se han elegido las plataformas a utilizar para el sistema. Existen diversos lenguajes y entornos de programación para cada una de estas plataformas. Entre todos ellos se ha decidido utilizar para el desarrollo de esta aplicación el lenguaje de programación java por ser viable tanto para la programación de los Arduino como para la programación sobre Raspberry Pi.

4.2.1. Elección del entorno de desarrollo para el gestor de los nodos de la red de sensores

Se utilizará el lenguaje de programación java y la interfaz de desarrollo para Arduinos IDE Arduino-1.0.5.

El entorno de desarrollo en java para Arduinos es un entorno completo y estable. Dispone de un API completa y bien documentada, y es posible encontrar gran cantidad de ejemplos.
[16]

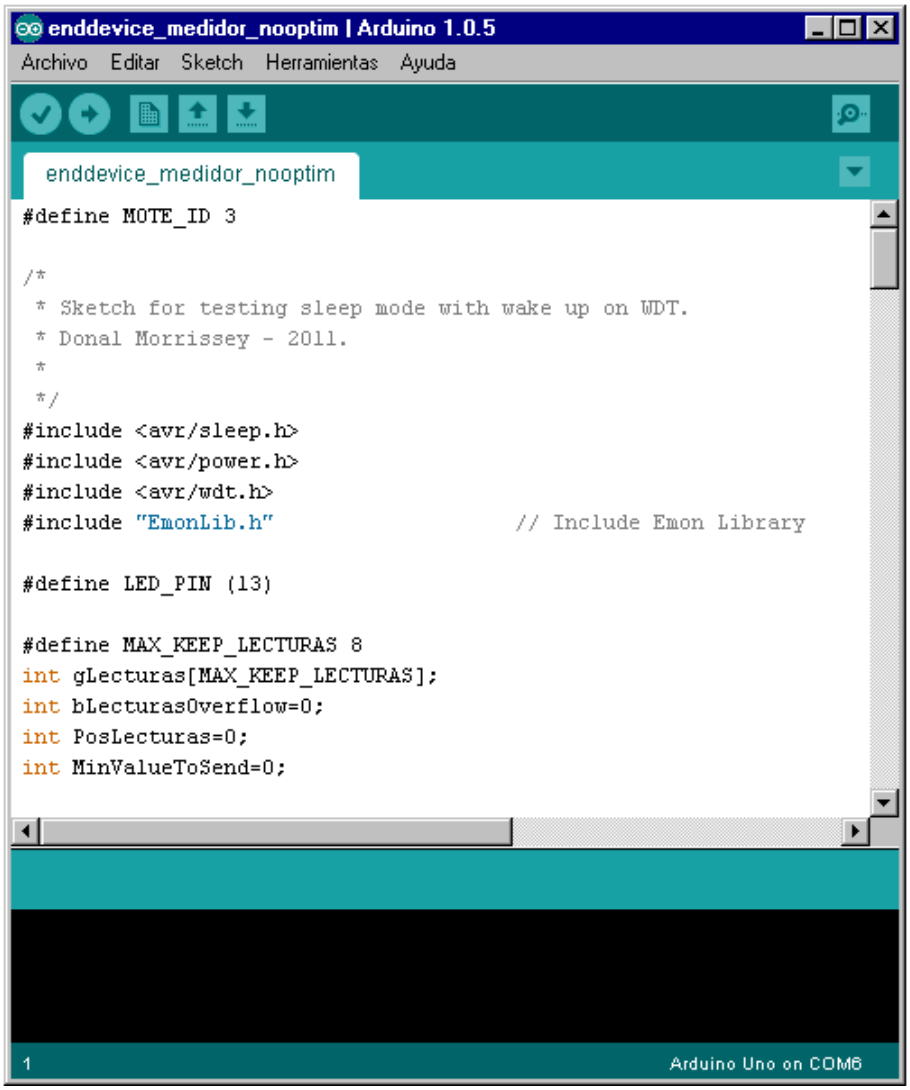

Figura 25: IDE Arduino-1.0.5

4.2.2. Elección del entorno de desarrollo para el gestor del centro de control

Para la aplicación a desarrollar para Raspberry Pi también se utilizará java. Raspberry Pi ejecuta Raspbian, una versión derivada de Debian, que lleva instalada una versión completa de la máquina virtual de Java. Por lo tanto el desarrollo de esta parte de la aplicación no debe suponer ningún problema y es posible utilizar cualquier de los entornos de programación para Java existentes.

En concreto se ha utilizado la interfaz de desarrollo Eclipse. Se ha elegido esta interfaz por ser una de las interfaces de desarrollo para Java más avanzada y con más facilidades, y especialmente por facilitar enormemente el diseño de interfaces gráficas basadas en swing.

Concretamente se ha utilizado la versión Eclipse Java EE IDE Kepler *Service Release* 2.

4.3. Diseño del protocolo de comunicación

Como primer paso antes de comenzar la codificación de cada parte del sistema se debe diseñar el protocolo de comunicación entre el coordinador de la red y cada nodo medidor.

4.3.1. Caso de uso de los nodos medidores

Es preferible buscar la simplicidad en los nodos terminales. Estos nodos suelen ser dispositivos con una capacidad limitada, porque lo que se debe encontrar un compromiso entre la optimización y la simplicidad.

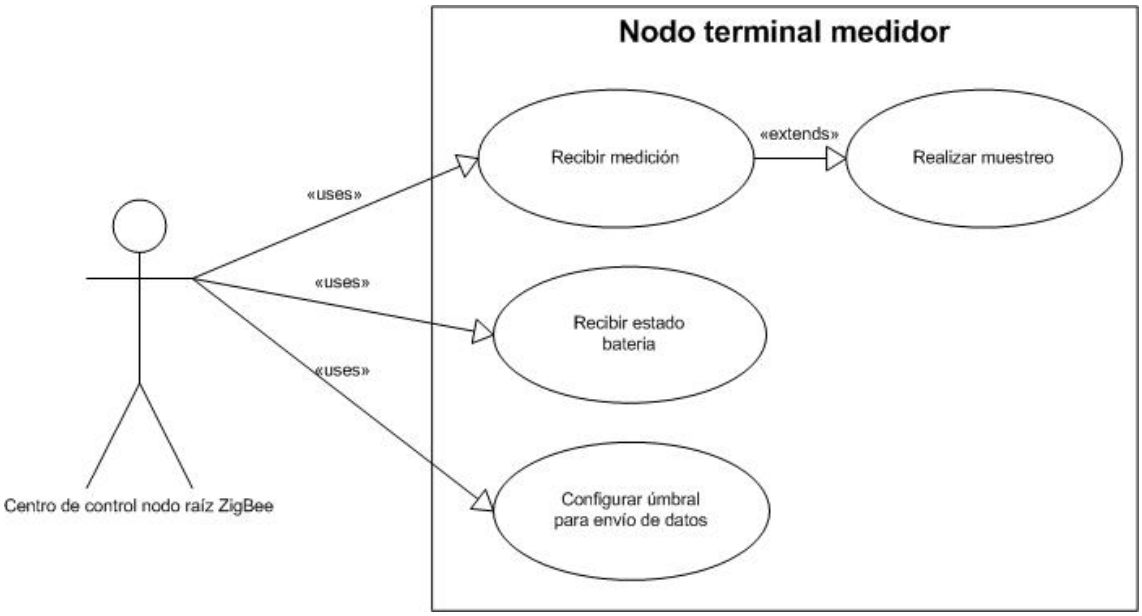

Figura 26: Caso de uso nodos medidores

En apartados posteriores se profundizará en las optimizaciones implementadas y el funcionamiento interno de los nodos terminales de la red. En este apartado, para diseñar el protocolo de comunicación de la red ZigBee, solamente se necesita tener claro la funcionalidad que cada nodo terminal exporta hacia el resto de la red. En este caso, puesto que se trabaja con una topología en forma de estrella, la funcionalidad que exporta hacia el nodo coordinador de la red (los nodos terminales no se comunican entre ellos). A continuación se muestra el caso de uso de cada nodo terminal por parte del nodo coordinador de la red ZigBee.

4.3.2. Solicitudes y respuestas

A continuación se describen las posibles solicitudes/respuestas que pueden intercambiar el coordinador de la red ZigBee con los nodos medidores.

4.3.2.1. Recibir medición

Para solicitar un envío de medición de corriente eléctrica el coordinador difunde en *broadcast* en la red, hacia todos los nodos medidores, un mensaje con el valor 0x21 (“!”). Los nodos que escuchan dicho mensaje contestan con un paquete como el que se describe a continuación donde se devuelve la medición junto con un *timestamp* y el id del nodo que envía la respuesta.

Se debe tener en cuenta que los dispositivos Arduino no disponen de un reloj interno permanente, por lo tanto cada vez que el dispositivo es reseteado, el contador vuelve a cero de nuevo. El *timestamp* que se envía es el número de milisegundos desde que se inició el dispositivo por última vez.

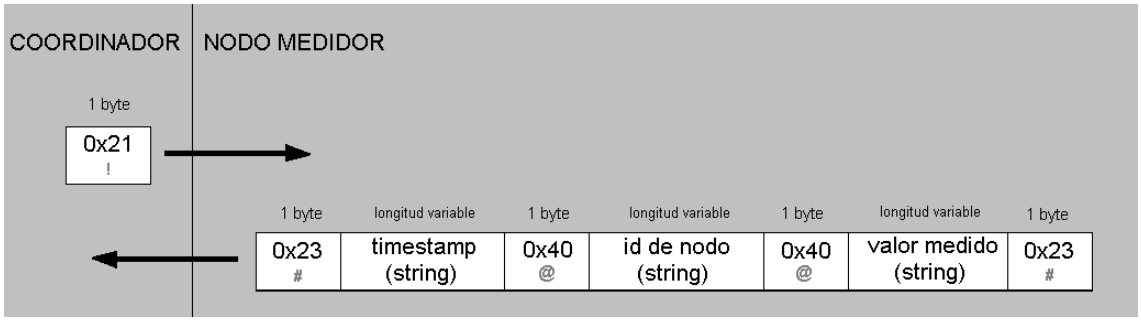

Figura 27: Petición/respuesta recibir medición

4.3.2.2. Configurar umbral mínimo para envío de datos

Una de las optimizaciones implementadas es la posibilidad de configurar a los nodos un umbral mínimo para el envío de datos. Es decir, solamente los valores que superen dicho valor, serán enviados.

Esta optimización se inspira en los modificadores a los SELECT de TinyDB que permiten mandar parte del filtrado de información a los nodos medidores.

Para configurar un valor mínimo a los nodos se envía el valor 0x2f (“/”) seguido por el umbral a configurar en formato texto. La respuesta se puede ver en el paquete que se describe a continuación.

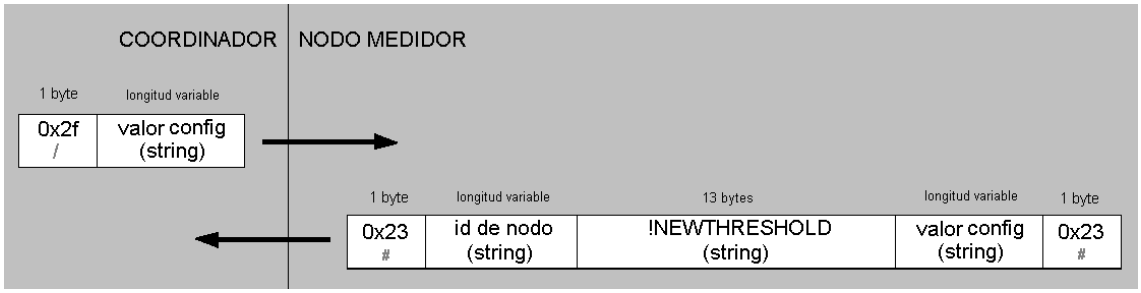

Figura 28: Petición/respuesta configuración de umbral mínimo

4.3.2.3. Notificación batería baja

En este caso no es necesario solicitar mediante un comando el envío de la notificación de batería baja. Todo nodo que en un momento dado detecte un nivel de voltaje de batería inferior a 4 voltios, enviará el paquete de notificación que se describe a continuación.

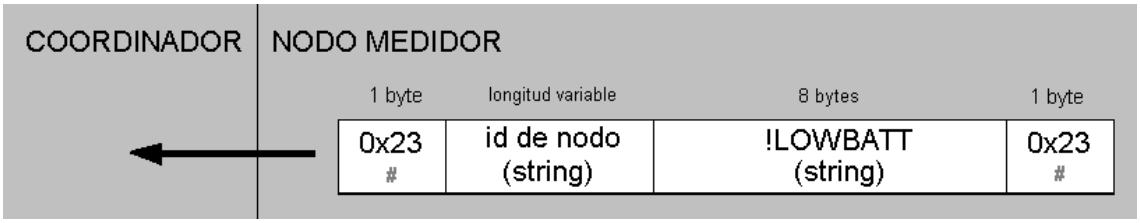

Figura 29: Petición/respuesta notificación batería baja

4.4. Diseño de la aplicación para el coordinador de la red de sensores

El código de la aplicación que se cargará en el coordinador de la red de sensores será bastante sencillo. El coordinador es un mero conector del centro de control principal con la red de sensores, por lo que su función será principalmente reenviar las mediciones recibidas desde la red de sensores hacia el centro de control principal.

Esta sencillez de código se debe en parte al módulo Xbee Series 2, que nos abstrae de toda la gestión de la red ZigBee y nos permite tratar con la red simplemente a través del puerto serie de Arduino.

Sin embargo todavía se necesita resolver algunos problemas para su correcto funcionamiento.

Por un lado, aunque el módulo Xbee nos abstrae de la gestión de la red, se debe configurar correctamente dicho módulo con los parámetros adecuados. Por otra parte tanto el módulo Xbee Series 2 como el puerto USB de los Arduino funcionan a través del puerto serie. El coordinador de la red debe recibir datos mediante Xbee Series 2 para reenviar a través del puerto USB, así que se deberá resolver este conflicto. En el apartado sobre detalles de implementación se estudiará cómo solucionar cado uno estos problemas.

4.4.1. Diagrama de flujo coordinador de la red de sensores

A continuación se muestra el diagrama de flujo de la aplicación a cargar en el Arduino coordinador de la red ZigBee.

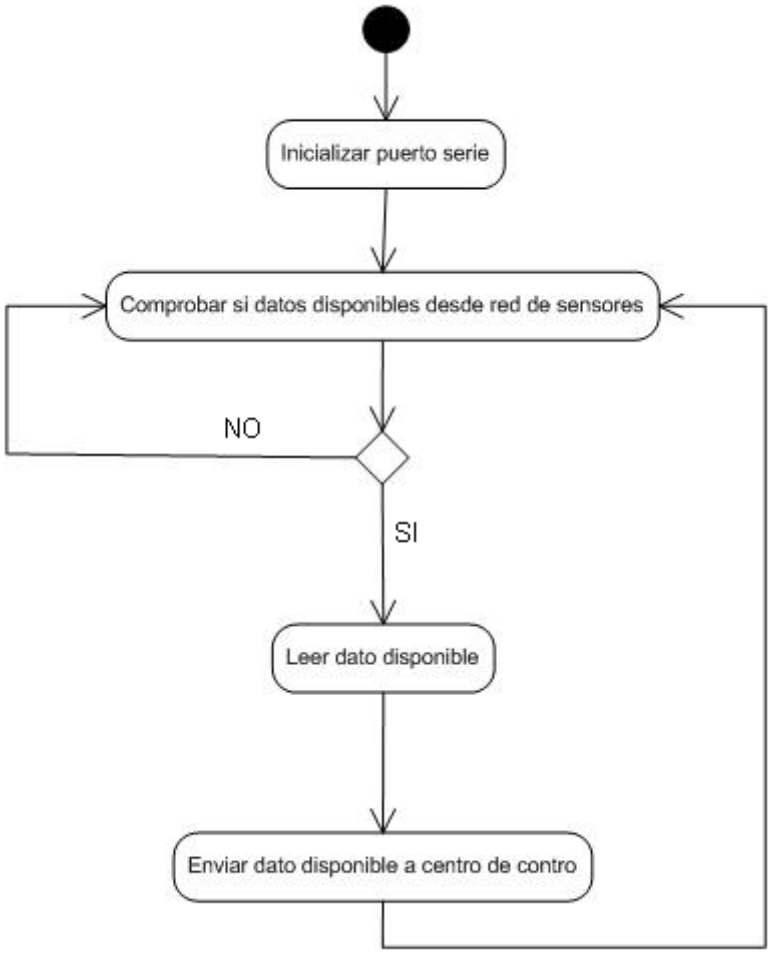

Figura 30: Diagrama de flujo del coordinador de la red ZigBee

4.5. Diseño de la aplicación para cada nodo medidor

El código de la aplicación que se cargara en cada nodo medidor no es de gran complejidad, sin embargo hay bastantes más detalles de implementación que deben ser resueltos. Se verá más adelante en apartados dedicados a los detalles de implementación.

En la aplicación para cada nodo servidor se han implementado una serie de optimizaciones inspiradas en la plataforma TinyDB que permitirán reducir el consumo de energía por parte de

los nodos medidores así como reducir el tráfico de red. Se hablará de cada una de ellas a continuación.

En cuanto al resto de la aplicación, su principal función será quedar a la escucha de peticiones recibidas desde el coordinador para contestar con la información adecuada.

4.5.1. Optimizaciones en la aplicación de los nodos medidores

A continuación se describirán las optimizaciones que serán implementadas en el sistema de monitorización de consumo eléctrico.

4.5.1.1. EPOCH (*once per time-period*)

Una de las optimizaciones que se implementarán se basa en el uso de EPOCH de TinyDB. Un EPOCH es un intervalo de muestreo. Todos los nodos duermen al mismo tiempo y todos despiertan a la vez para volver a formar la red ZigBee, escuchar peticiones, realizar muestreos, enviar las respuestas adecuadas, etc… De esta manera, utilizando EPOCH, la mayor parte del tiempo los nodos duermen, con el consecuente ahorro de energía.

Arduino permite desde código hacer entrar a la placa en distintos modos de *standby*. Cada uno de ellos desactiva distintas partes de la placa. Esto se verá en detalle en los apartados de detalles de implementación. Para esta aplicación caso es posible entrar en el llamado modo *power down*, uno de los modos de máximo ahorro en el que se desactiva casi todo mientras el nodo duerme.

Se implementará un EPOCH de 8 segundos (aunque cuando se detecta batería baja, este tiempo puede bajar hasta 24 segundos).

El coordinador no entra en nunca estado de dormido, siempre está activo (no hay problemas de energía ya que está conectado al centro de control y en principio no es tan crítico el ahorro de energía en esta parte).

4.5.1.2. Almacenamiento de datos en la red

Otra de las optimizaciones que se implementará es el almacenamiento de datos en los propios nodos medidores. Esta optimización también está inspirada en TinyDB, concretamente en sus "*storage points*" o puntos de almacenamiento.
[17]

En un esquema más sencillo se enviaría cada muestreo de sensores hacia el coordinador poco después de realizarse dicho muestreo. Pero esto puede derivar en una pérdida de rendimiento y en un consumo excesivo de batería si en ese momento el coordinador no escucha o no interesan los datos capturados.

Por tanto cuando un nodo despierta, es mucho más interesante que el nodo realice el muestreo de sensores, y lo guarde en un *buffer* (una caché) en el propio nodo.

En la implementación a realizar esta caché tendrá capacidad para ocho lecturas de sensores. A partir de la octava lectura no enviada, se empezarán a sobrescribir los datos más antiguos.

Si durante un EPOCH el nodo recibe solicitud de datos desde el coordinador, enviará todos los datos en la caché pendientes de ser enviados.

4.5.1.3. Esquema de pila *naive*

Una vez más el diseño de la aplicación se inspira en TinyDB, concretamente en los esquemas de almacenamiento de datos propuestos. Para el almacenamiento de datos en los nodos medidores se utilizará un esquema sencillo en el que los datos más antiguos del buffer son los primeros que se enviarán al coordinador. Además según se va recibiendo datos nuevos, se sobrescribirá los más antiguos con estos datos más nuevos. En TinyDB esto se conoce como esquema de pila *naive*.

4.5.1.4. Filtrado de datos realizado directamente en los nodos

En TinyDB es posible añadir modificadores a las *queries*, por ejemplo a SELECT, para que se realicen una serie de operaciones sobre los datos directamente en los nodos. Aquí se ve un ejemplo:

SELECT * FROM sensors AS s, recentLight AS rl

```
WHERE  rl.nodeid = s.nodeid
AND  s.light < rl.light
   SAMPLE PERIOD 10s
```

Como se puede observar, la *query* anterior solicita que se envíen los muestreos de luz cuyo valor supere un umbral previo. De esta manera solo se envía hacia el servidor los datos interesantes, descartando el resto, con el consecuente ahorro en tráfico de red y ahorro de energía derivado de no hacer uso del sistema de radio.

En la aplicación medidora de consumo eléctrico también se implementara una versión simplificada de un filtro de datos ejecutable en los nodos para filtrar que valores se enviarán y cuáles no. Este filtro consiste en un umbral configurable mediante un comando desde el coordinador, como ya se vio en el diseño del protocolo de comunicación. Cuando se inicia el nodo, el umbral es cero (todos los valores se guardan para ser enviados). Cuando este valor se configura, solo se guardan las lecturas de los sensores superiores al umbral configurado.

4.5.1.5. *Lifetime*

Al describir TinyDB ya se comentó sobre el modificador *lifetime*, que sirve para especificar que el período de muestreo garantice un tiempo de vida (de batería) mínimo para el nodo. Por ejemplo: Select nodeid, accel from sensors lifetime 30 days, dice al nodo que muestree al máximo posible, pero garantizando un tiempo de vida de 30 días.

Las placas Arduino ofrecen otra facilidad útil para la aplicación. Se puede conocer el voltaje de entrada de la placa y con ello conocer aproximadamente el estado de carga la batería. Una caída en el voltaje probablemente indique que la batería que mantiene la placa activa está agotándose.

Cuando se detecte dicha situación, se realizarán lecturas y envíos cada tres EPOCH, es decir, cada 24 segundos, de manera que el nodo siga funcionando aunque realice muestreos cada más tiempo. Además cuando se detecte batería baja, se notificará al coordinador mediante el mensaje descrito en el apartado de diseño del protocolo de comunicaciones.

4.5.2. Diagrama de flujo de la aplicación de los nodos medidores

A continuación se muestra el diagrama de flujo principal de la aplicación que se cargará en los nodos medidores.

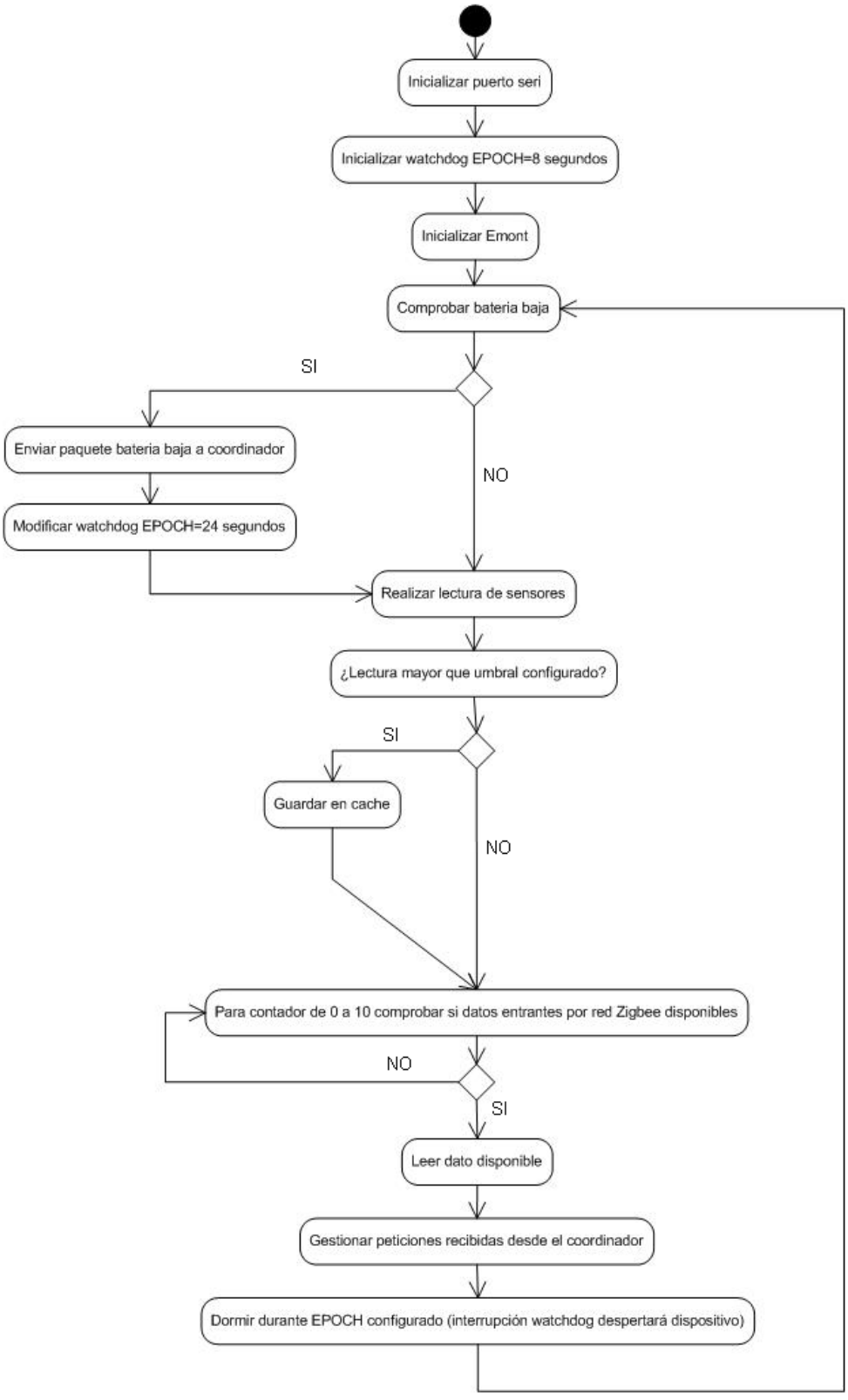

Figura 31: Diagrama de flujo principal medidor

A continuación se muestra el diagrama de flujo de la gestión de peticiones recibidas desde el coordinador de la red ZigBee:

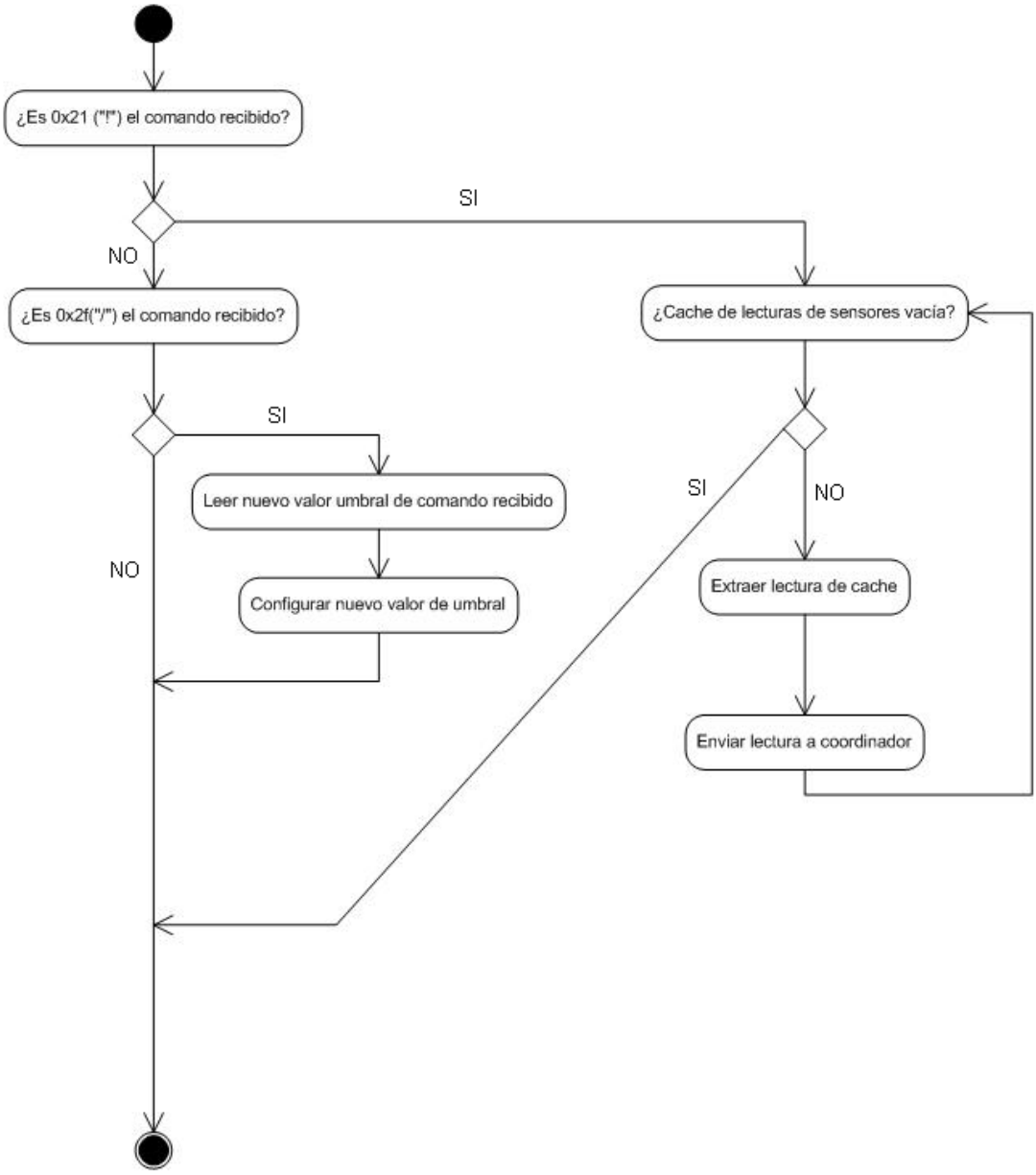

Figura 32: Diagrama de flujo medidor gestión de comandos

4.6. Diseño de la aplicación para el centro de control principal

El centro de control principal recibe la información de la red de medidores a través del nodo coordinador de la misma, el Arduino que actúa como coordinador, y que forma parte del

centro de control. Dicho Arduino se conecta al centro de control principal (el cual se ejecuta sobre la Raspberry Pi) mediante un cable USB.

La aplicación a ejecutarse sobre la Raspberry Pi no tiene las limitaciones que tenían los programas cargados en los Arduino. Los dispositivos Arduino son de una capacidad muy inferior a las Raspberry Pi y no están pensados para ejecutar grandes aplicaciones.

Además Raspberry Pi ejecuta Raspbian, una versión derivada de Debian, por lo que ofrece todo tipo de herramientas y librerías como cualquier sistema operativo moderno y completo de propósito general.

La aplicación que ejecutará el centro de control del sistema estará desarrollada en Java. Dicha aplicación ofrecerá una interfaz gráfica al usuario autenticado localmente en la Raspberry Pi. Está aplicación además se comunicará con el coordinador de la red ZigBee de medidores y recibirá de la misma las mediciones enviadas por los dispositivos, y se las mostrará al usuario en pantalla.

Como ya se vio en el caso de uso de la figura 13, el usuario tendrá la posibilidad de comenzar y parar la captura de datos recibidos desde la red de medidores.

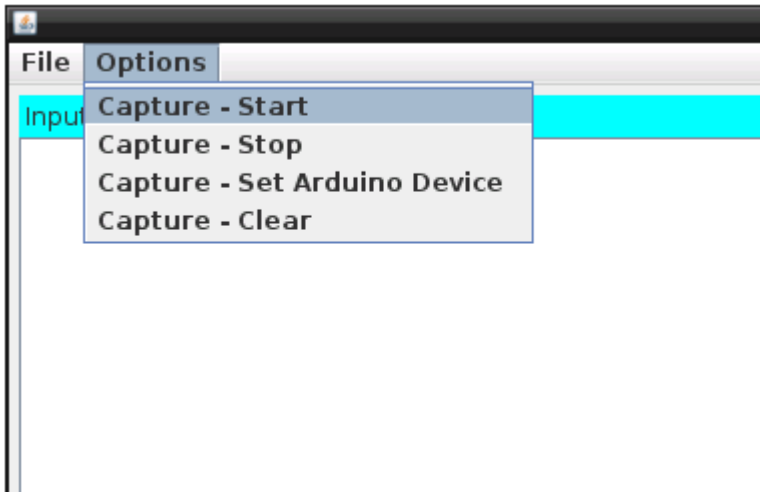

Figura 33: Centro de control opciones

La aplicación mostrará, sobre cada captura, la hora de llegada de la información al centro de control, el valor de la medición, el id del nodo que hizo la medición y los milisegundos desde que se inició por última vez el Arduino del nodo que hizo dicha medición hasta que se hizo la medición.

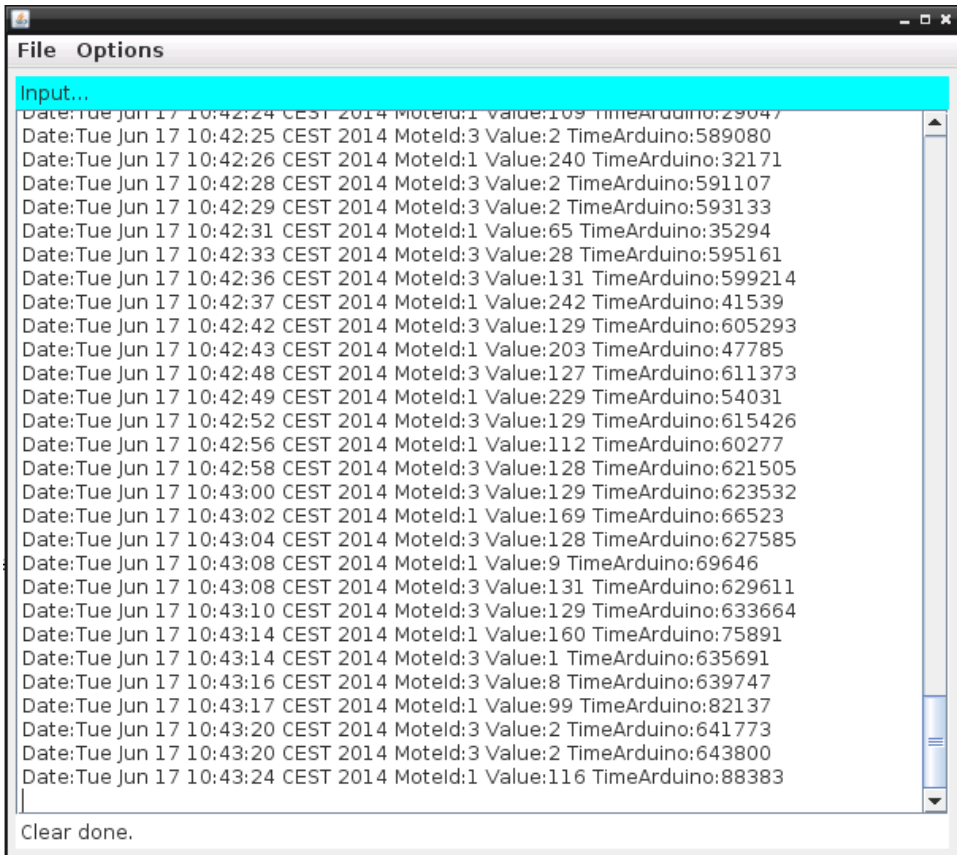

Figura 34: Centro de control muestra de datos recibidos

El usuario también tendrá la posibilidad de salvar o cargar datos de una sencilla base de datos implementada sobre ficheros en disco.

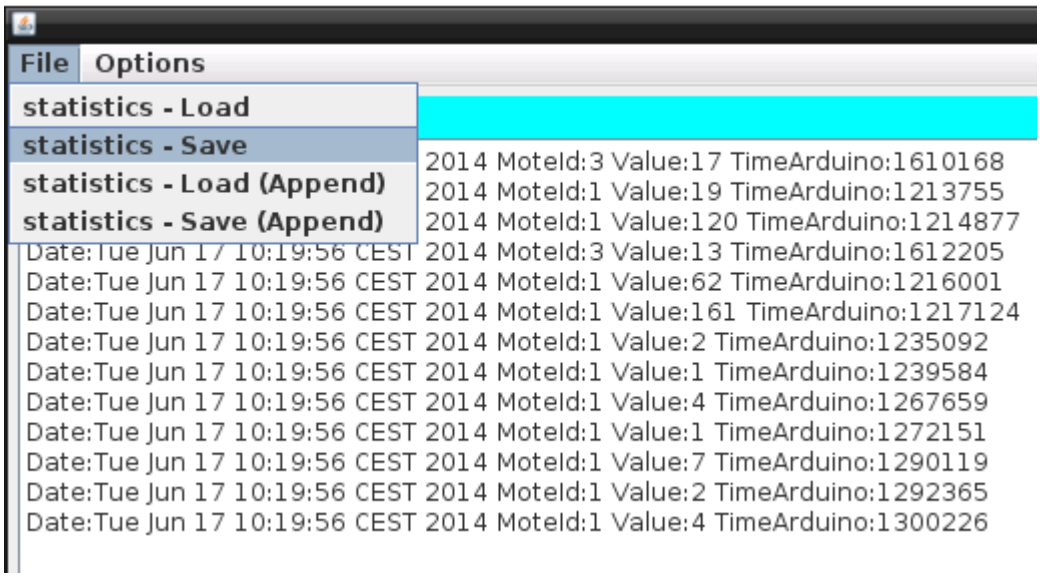

Figura 35: Centro de control muestra de datos recibidos

4.6.1. Diagrama de clases del centro de control

A continuación se muestra un diseño de alto nivel de las clases que compondrán la aplicación en Java del centro de control. Este diseño es una idea general y orientativa que no se corresponde exactamente con las clases en código pero que facilita la comprensión de la aplicación.

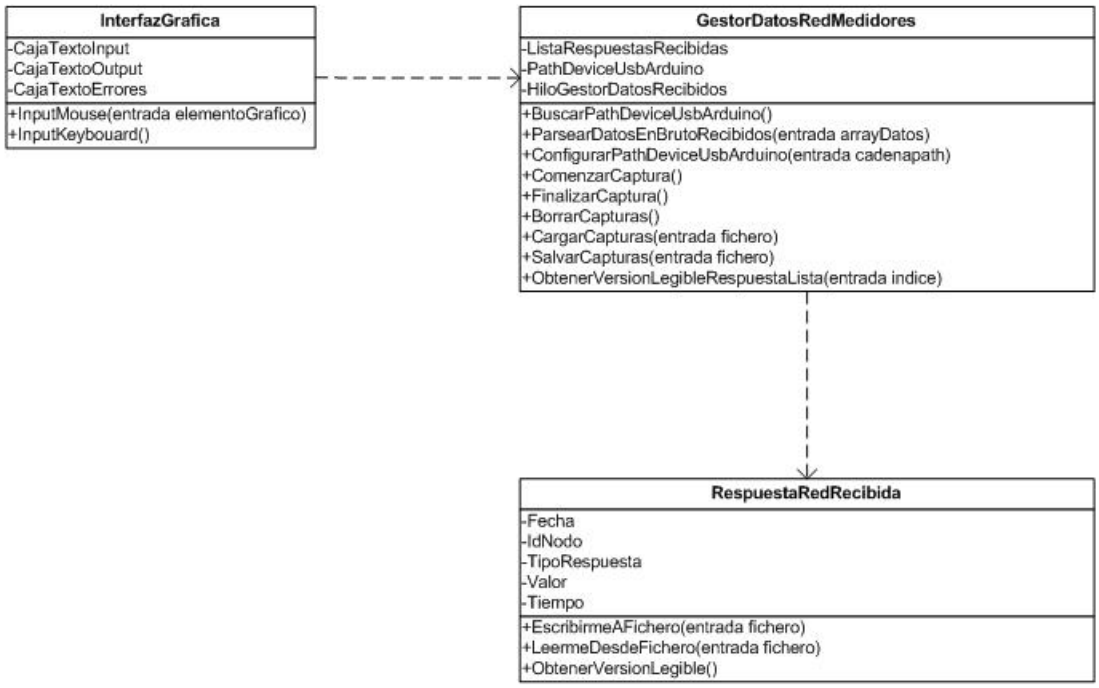

Figura 36: Diagrama de clases del centro de control

Capítulo 5 Implementación de la aplicación

En los apartados anteriores se ha hecho un diseño a alto nivel de cada componente del sistema que se pretende desarrollar.

En los próximos apartados se hará un estudio más técnico en el que se describirán en profundidad los detalles de implementación más destacados, problemas encontrados y soluciones llevadas a cabo.

5.1. Detalles de implementación en el coordinador de la red de sensores

Como ya se vio en el apartado de diseño el código a cargar en el coordinador de la red de sensores es bastante sencillo, ya que su única función es reenviar la información recibida de los medidores hacia el centro de control principal que será quien se encargue de gestionar los datos.

Sin embargo es importante hablar sobre algunos problemas encontrados y las soluciones tomadas así como describir la manera de configurar los módulos Xbee Series 2 para actuar como coordinador.

5.1.1. Configuración del módulo Xbee Series 2 como coordinador

Para realizar la configuración de los módulos de radio de la firma Digi se debe utilizar el software multiplataforma desarrollado por la compañía, X-CTU. Esta herramienta ofrece todas las utilidades necesarias para cargar la configuración de red en los módulos Xbee conectados a los Arduino.
[18]

Es posible crear un enlace directo entre dos dispositivos Xbee enlazándolos mediante sus números de serie. En el sistema que se está implementando no tiene sentido tener un único medidor en la red, por lo que el coordinador de la red ZigBee actuará como coordinador en *broadcast,* y los nodos medidores se conectarán a dicho coordinador formando una topología en forma de estrella.

Para que cada dispositivo de la red quede identificado unívocamente, todos los dispositivos disponen de un número de serie, que les identificará y formará parte del direccionamiento de la red.

Por otro lado, para que los dispositivos de la misma red sepan a qué red pertenecen, se comuniquen con otros dispositivos de la misma red, e ignoren dispositivos de otras redes evitando que haya interferencias entre ambas, se utiliza el llamado PAN ID (*personal area network identifier*). La red de nuestro sistema será configura con el PAN ID 3331.

A continuación se describe la configuración de red necesaria para el coordinador.

Para comunicarnos con el módulo Xbee se debe utilizar una velocidad de transmisión de 9600 *bits* por segundo, control de flujo hardware, símbolos de 8 bits con 1 *bit* de parada y control de flujo por hardware.

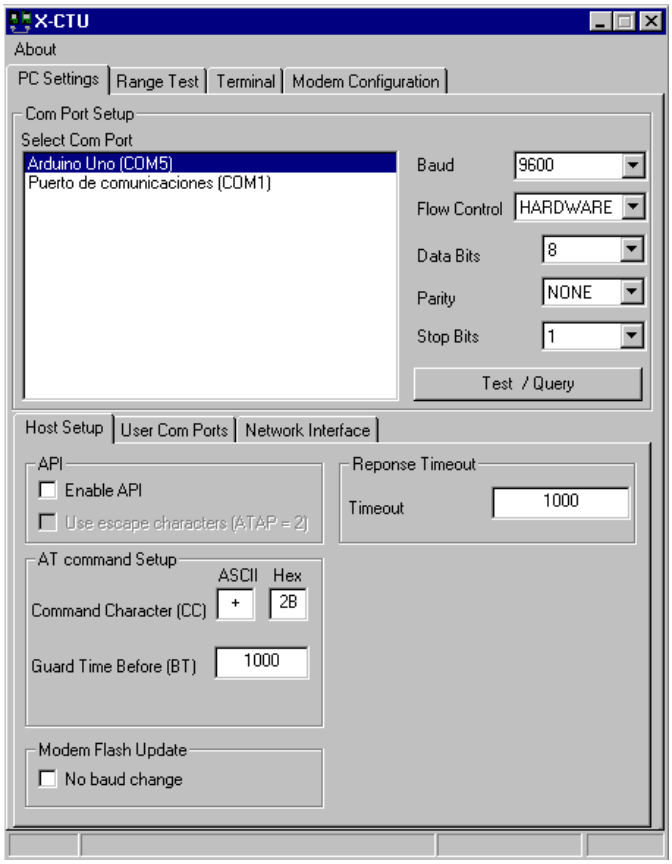

Figura 37: Configuración conexión X-CTU

Seleccionando el puerto COM al que esté conectado el dispositivo Arduino funcionando como coordinador es posible obtener el número de serie del módulo Xbee conectado a dicho

dispositivo pulsando el botón "*Test/Query*". Este número de serie es único para este dispositivo y le identifica unívocamente en la red.

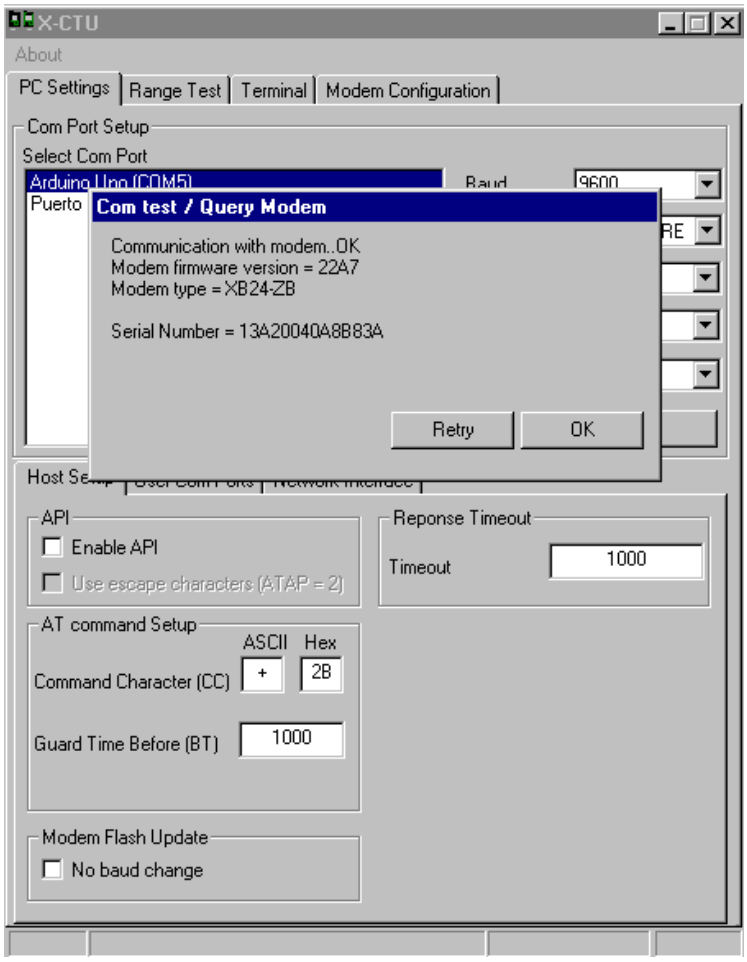

Figura 38: Número de serie Xbee Series 2 con X-CTU

Para introducir la configuración de red adecuada para el coordinador del sistema de medidores es necesario ir a la pestaña "*modem configuration*".

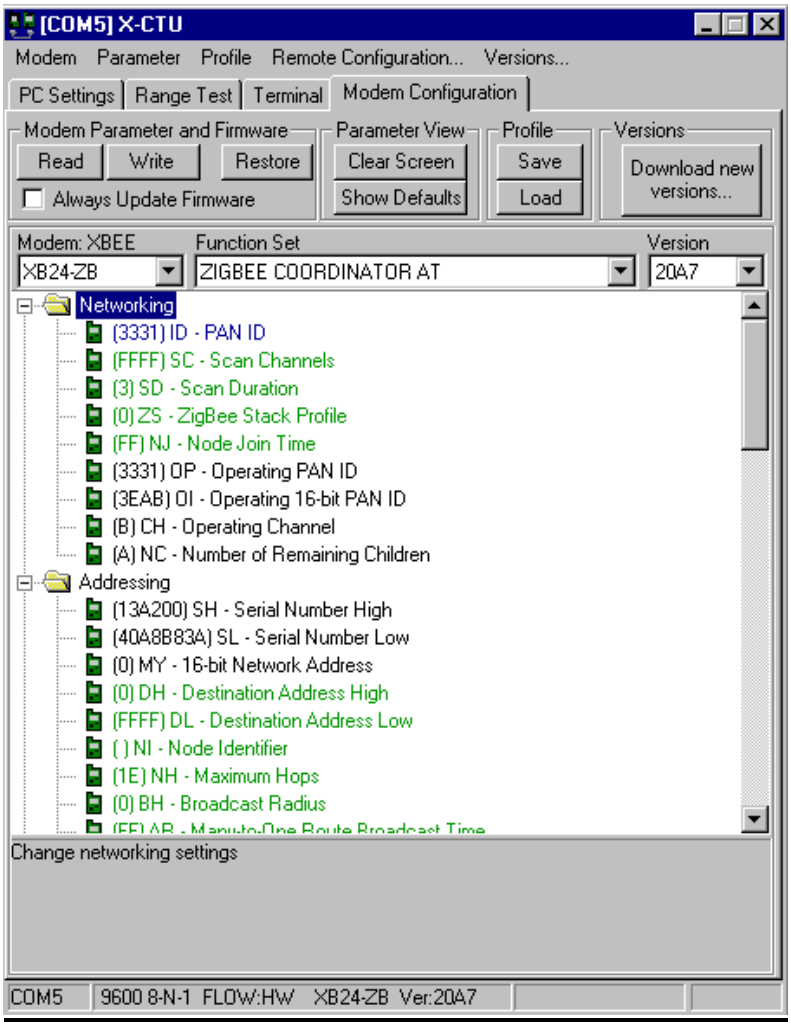

Figura 39: Configuración del *modem* del coordinador *Zigbee*

En dicha pestaña se debe seleccionar la función “ZIGBEE COORDINATOR AT”, ya que este dispositivo será el coordinador de la red.

El parámetro más importante es el PAN ID. El PAN ID es el identificador de red para la red ZigBee siendo configurada. La red ZigBee de nuestro sistema utilizará el PAN ID 3331. Los nodos con diferente PAN ID se ignorarán entre ellos.

El número de serie también es importante, pues se utilizará durante el direccionamiento de red como dirección única que identifica al nodo. Va puesto automáticamente al número de serie de este módulo.

En el apartado de direccionamiento se ven también los campos “*Destination Address High*” y “*Destination Address Low*”. En estos campos se debe introducir el número de serie del nodo destino con el que se quiere enlazar. En el caso del coordinador del sistema se debe asignar el valor 0x000000000000ffff como dirección destino. Esta es la dirección *broadcast*. Mediante

la asignación de este valor se pone al coordinador a funcionar en modo *broadcast* permitiéndole comunicarse con todos los dispositivos finales de la red del mismo PAN ID.

Los dispositivos finales sin embargo deben ser configurados con el número serie del coordinador como dirección destino, aunque esto se verá cuando se analicen los detalles de implementación de los medidores.

5.1.2. Solución al conflicto del puerto serie: USB y Xbee Series 2

Ya se comentó cuando se describió los módulos Xbee Series 2 que dichos módulos funcionan de manera totalmente autónoma en lo que respecta a gestión de red y que abstraen totalmente al programa ejecutándose en el Arduino de dicha gestión, ofreciendo una sencilla interfaz a través del puerto serie.

Cuando desde el código ejecutándose sobre Arduino se realiza un envío de datos al puerto serie estándar, los datos llegan al módulo Xbee Series 2 y, según la configuración del mismo, estos datos son enviados a la red de una manera o de otra. En el caso del coordinador los datos serán difundidos en *broadcast* en la red, llegando a todos los dispositivos finales.

Sin embargo en el coordinador se encontró el problema de que tanto el módulo Xbee como el puerto USB funcionan a través del puerto serie de Arduino. Y el coordinador de la red por un lado está conectado a la red ZigBee a través del módulo Xbee y por otro lado está conectado al centro de control principal mediante USB.

El problema se soluciona gracias a la placa Xbee Shield Pro que hace de intermediaria entre la placa Arduino y el módulo Xbee Series 2. En dicha placa se encuentran dos *jumpers* que determinan como interactúa el microcontrolador del módulo Xbee con el chip serie FTDI de Arduino.

En la posición Xbee el pin DOUT del módulo Xbee conecta al pin RX del microcontrolador y el pin DIN a TX. A su vez los pines TX y RX aún están conectados a TX y RX del chip FTDI, por lo tanto los datos enviados desde el microcontrolador del módulo Xbee son enviados vía USB y a la vez enviados a la red ZigBee inalámbricamente. Si embargo, el microcontrolador solo será capaz de recibir datos desde el módulo Xbee, no desde USB. Con los *jumpers* en la posición USB, Arduino solo será capaz de comunicarse con el ordenador vía USB. En ocasiones se ha podido observar que es necesario desconectar el módulo Xbee para una correcta comunicación por USB.
[12]

De esta manera es posible solucionar el problema del uso de Xbee Series 2 y a la vez el puerto USB. Desde Arduino es posible enviar y recibir de la red, y con la configuración de *jumpers* adecuada, además reproducir en USB lo enviado. De esta manera los datos enviados irán tanto al puerto USB (al centro de control) como a los dispositivos medidores, pero gracias al protocolo definido, cada receptor filtrará los mensajes recibidos quedándose solamente con los paquetes que le interesan. Los nodos medidores gestionarán las peticiones realizadas por el coordinador y el centro de control gestionará las respuestas recibidas desde la red ZigBee reenviadas por el coordinador a través del puerto USB.

Como contrapartida se tiene el problema de que para que el coordinador sea capaz de reenviar las respuesta de la red ZigBee hacia el centro de control a través del cable USB, el programa ejecutándose en Arduino debe enviar dichas respuestas al puerto serie estándar, compartido por el módulo Xbee Series 2 y USB, por lo que en la red ZigBee se podrá observar un eco por parte del coordinador. Esto no es mayor problema porque las respuestas simplemente serán ignoradas por los dispositivos finales (incluso posiblemente ya estén en el período de *standby* del EPOCH en ese momento).

5.1.3. Código en java para el Arduino coordinador

Como ya se comentó en el apartado de diseño el coordinador simplemente reenviará las respuestas recibidas desde la red ZigBee hacia el cable USB utilizando tanto para recibir como para enviar el puerto serie estándar de Arduino. Esta parte del sistema no tiene mayor complicación, salvo por los detalles comentados en los dos apartados anteriores.

5.2. Detalles de implementación en la aplicación de los nodos medidores

A continuación describimos los detalles de implementación más destacados y problemas encontrados durante la implementación de los nodos medidores.

5.2.1. Configuración del módulo Xbee Series 2 como dispositivo final

Puesto que ya se ha descrito la aplicación X-CTU en apartados anteriores no será necesario profundizar. Simplemente hay que tener en cuenta que para configurar correctamente cada dispositivo final medidor de la red ZigBee se debe configurar el mismo PAN ID que se

configuró en el coordinador de dicha red y se debe configurar como dirección de destino el número de serie del coordinador.

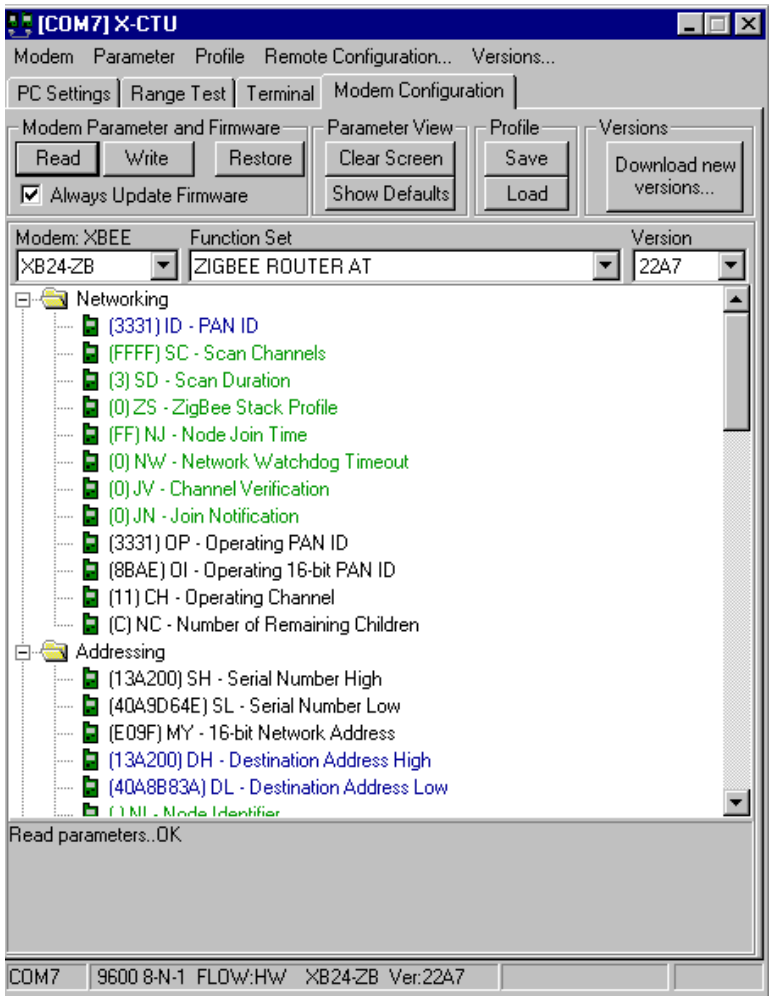

Figura 40: Configuración del *modem* de los dispositivos finales Zigbee

5.2.2. Detalles de implementación del "*Once Per-Time Period*" EPOCH

En el diseño ya se comentó sobre la optimización implementada inspirada en el uso de EPOCHs de TinyDB. Un EPOCH es un intervalo de muestreo. Los nodos de la red duermen la mayor parte del tiempo y despiertan para realizar las tareas de muestreo, recepción de peticiones desde el coordinador, etc…

De esta manera, utilizando EPOCH, la mayor parte del tiempo los nodos duermen, con el consecuente ahorro de energía.

Arduino nos permite desde código hacer entrar a la placa en distintos modos *de standby*. Cada uno de ellos desactiva distintas partes de la placa:

SLEEP_MODE_IDLE	Menor ahorro de energía.
SLEEP_MODE_ADC	
SLEEP_MODE_PWR_SAVE	
SLEEP_MODE_STANDBY	
SLEEP_MODE_PWR_DOWN	Mayor ahorro de energía.

Tabla 4: modos *standby* de Arduino

En la aplicación desarrollada es posible utilizar el modo que más ahorro de energía permite: SLEEP_MODE_PWR_DOWN. En este modo se desactiva prácticamente por completo en la placa Arduino.

Hay diferentes métodos para volver a despertar la placa Arduino:

- Interrupción externa (cambios en los pins configurados).
- Hardware UART (interfaz serie).
- Timer interno o *Watchdog Timer*.

En el caso del modo SLEEP_MODE_PWR_DOWN la UART duerme también, por lo que mientras el modo duerme todo lo que se recibe por serie (y por tanto todo lo que llega por radio por el módulo Xbee) se pierde. Además no es posible utilizar la UART como *trigger* para despertar la placa en este modo. En modos menos restrictivos como el modo SLEEP_MODE_IDLE, la UART no se desactiva, por lo que no se perdería información mientras se está en estado *standby*, pero el ahorro de energía quedaría lejos del conseguido gracias al modo SLEEP_MODE_PWR_DOWN.

Para poder utilizar el modo SLEEP_MODE_PWR_DOWN se utiliza la siguiente técnica. Desde el coordinador se envían continuamente las peticiones y solicitudes hasta recibir respuesta. Si desde el coordinador se envían peticiones durante al menos lo que dura un EPOCH, queda asegurada la recepción de las peticiones por parte de los nodos terminales de la red.

Esto no supone ninguna pérdida de rendimiento o gasto energético. El coordinador está conectado al centro de control y no es necesario optimizar en ahorro energético en este nodo, por lo que no implica una pérdida de rendimiento la difusión continua de peticiones a la red.

En esta aplicación se ha configurado un EPOCH de 8 segundos, aunque cuando se detecta la situación de batería baja, este tiempo puede incrementarse hasta 24 segundos.

El coordinador no entra en estado de *standby*, siempre está activo (no hay problemas de energía ya que está conectado al centro de control).

Para configurar el modo *sleep*, se debe previamente configurar la interrupción del *Watchdog Timer* que nos permitirá salir del *sleep* cada 8 segundos, hacer los muestreos de sensores y envíos por radio pertinentes, y volver a *sleep*.

A continuación se muestran los valores de *prescaler* configurables para distintos períodos de *timeout* del *watchdog*.

WDP3	**WDP2**	**WDP1**	**WDP0**	**Ciclos de oscilación del WDT (ciclos)**	**Timeout**
0	0	0	0	2048	16 ms
0	0	0	1	4096	32 ms
0	0	1	0	8192	64 ms
0	0	1	1	16384	0.125 s
0	1	0	0	32768	0.25 s
0	1	0	1	65536	0.5 s
0	1	1	0	131072	1 s
0	1	1	1	262144	2 s
1	0	0	0	524288	4 s
1	0	0	1	1048576	8 s
1	0	1	0	Reservado	Reservado
1	0	1	1	Reservado	Reservado
1	1	0	0	Reservado	Reservado
1	1	0	1	Reservado	Reservado
1	1	1	0	Reservado	Reservado
0	1	1	1	Reservado	Reservado

Tabla 5: tabla de valores configurables para watchdog timer

Y a continuación el código necesario para realizar la configuración:

```
// Borrar flag de reset
MCUSR &= ~(1<<WDRF);

// Configurar WDCE para cambiar el prescaler
WDTCSR |= (1<<WDCE) | (1<<WDE);

// configurar el prescaler de timeout del watchdog a 8.0 segundos
WDTCSR = (1<<WDP3) | (0<<WDP2) | (0<<WDP1) | (1<<WDP0);

// Habilitar la interrupcion del watchdog
WDTCSR |= _BV(WDIE);
```

Tabla 6: configuración watchdog timer (código)

Para entrar al modo sleep se utiliza el siguiente código:

```
void enterSleep(void)
{
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);

  sleep_enable();

  // Entrar a modo sleep hasta que el watchdog
  // timer despierte al nodo
  sleep_mode();

  // El programa continua aqui despues del WDT timeout
  // Lo primero es deshabilitar el modo sleep
  sleep_disable();

  // Reactivar todos los perifericos
  power_all_enable();
}
```

Tabla 7: entrada de Arduino a modo standby (código)

5.2.3. Detalles de implementación del *lifetime*

Se hará continuación se hace una breve descripción también sobre la implementación de esta optimización por tener algunas peculiaridades en cuanto al control de estado de la batería del dispositivo. A continuación se describe la función con la que se lee el voltaje ofrecido por la batería.

```
long readVcc()
{
  // Read 1.1V reference against AVcc
  // set the reference to Vcc and the measurement to the internal 1.1V reference
  #if defined(__AVR_ATmega32U4__) ||
      defined(__AVR_ATmega1280__) ||
      defined(__AVR_ATmega2560__)
  ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  #elif defined (__AVR_ATtiny24__) ||
        defined(__AVR_ATtiny44__) ||
        defined(__AVR_ATtiny84__)
  ADMUX = _BV(MUX5) | _BV(MUX0);
  #elif defined (__AVR_ATtiny25__) ||
        defined(__AVR_ATtiny45__) ||
        defined(__AVR_ATtiny85__)
  ADMUX = _BV(MUX3) | _BV(MUX2);
  #else
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
  #endif
   delay(2); // Wait for Vref to settle
  ADCSRA |= _BV(ADSC); // Start conversion
  while (bit_is_set(ADCSRA,ADSC)); // measuring

  uint8_t low  = ADCL; // must read ADCL first - it then locks ADCH
  uint8_t high = ADCH; // unlocks both

  long result = (high<<8) | low;

  result = 1125300L / result; // Calculate Vcc (in mV); 1125300 = 1.1*1023*1000
  return result; // Vcc in millivolts
}
```

Tabla 8: Arduino lectura de voltaje de la batería (código)

Mediante este código es posible acceder al voltímetro no documentado de los Arduino 168 y 328.

[20]

Las versiones de Arduino 328 y 168 se han construido sobre una referencia de voltaje de 1.1 V. Es posible utilizar dicha referencia para medir el Vcc actual ofrecido por la batería. Para ello se utiliza el conversor analógico-digital para obtener la representación digital de Vcc y el valor de referencia. De esta manera, conociendo como es de grande la diferencia entre la representación digital de Vcc y del valor de referencia de 1.1 V, se puede conocer el valor analógico de Vcc.

Esta funcionalidad no está completamente documentada por la página oficial de Arduino por lo que no es posible asegurar el funcionamiento del código en futuros modelos de la placa.

5.2.4. Calibración del API de lectura de Emontx

Ya se comentó en apartados anteriores que el hardware a utilizar por los nodos medidores del sistema para realizar las mediciones de consumo eléctrico sería el sensor de corriente eléctrica no invasivo SCT-013, y que para facilitar la conexión del mismo a la placa Arduino, se utilizaría la placa, extensión de Arduino, Emontx.

Esta combinación de hardware es idónea para las necesidades de la aplicación en desarrollo, y Emontx facilita enormemente tratar con sensores como SCT-013 y otros de su familia.

Sin embargo es imposible fabricar algo con precisión absoluta, y en general, cuanta más precisión, más aumenta el precio.

Cuanto Emontx mide una corriente o un voltaje influyen varios factores en el resultado final, por ejemplo:

- La relación de transferencia del transformador de corriente.
- El valor de la resistencia de carga de Emontx.
- La precisión con la que se mide la tensión de la resistencia de carga.

En el momento de fabricación es imposible calibrar con total certeza el API de Emontx de manera que siempre mida con precisión en todos los entornos que sea usada. Puede estar siendo usada con diferentes versiones de Emontx (cada una con una resistencia de carga), diferentes sensores no invasivos, etc… [21]

Cuando se inicializa el API para Emontx desde el código cargado en la placa Arduino, se da una constante de calibración a dicho API. Mediante este parámetro es posible calibrar el valor de la medición devuelta por Emontx. [22]

```
EnergyMonitor emon1;                  // Crear instancia
…
emon1.current(1, 56.9);           // Configurar medición de corriente: (input pin, calibration cte).
```

Tabla 9: Emontx calibración (código)

El valor 56.9 es la constante de calibración, calculada como se describirá a continuación.

Para realizar una calibración fiable es necesario disponer de un polímetro con el que poder medir la corriente real atravesando el cable al que también está conectado nuestro sensor SCT-013.

Inicialmente, cuando el API de Emontx aún no está calibrado con el valor correcto, se le puede pasar cualquier valor como parámetro constante de calibración. Es recomendable utilizar el valor 111.1, que es el valor estándar dado por el fabricante OpenEnergyMonitor para una resistencia de carga de 18 ohmios.

Se debe conectar el sensor SCT-013 al cable sobre el que se hace la medición. Dicho sensor estará a su vez conectado a Emontx sobre Arduino. Y a su vez, el Arduino medidor debe estar conectado por ZigBee al coordinador, que recibirá las mediciones y las entregará al centro de control, donde será posible visualizarlas.

Con esta parte montada, el siguiente paso será conectar el polímetro al mismo cable sobre el que se están tomando las mediciones con SCT-013. El polímetro nos ofrece la posibilidad de conocer el valor real de corriente que está atravesando el cable, y de esta manera será posible comparar dicho valor real con la medición enviada por el Arduino medidor.

Lo ideal en este momento sería disponer de un generador de corriente alterna con un amplio rango de intensidades (se debe recordar que SCT-013 es capaz de medir corrientes alternas entre 50 mA y 100A). Sin embargo un generador de corriente de esos rangos es un instrumento caro.

Es posible realizar la medición sin necesidad del generador de corriente alterna. Para ello se utilizará una toma de corriente estándar de voltaje 220V. Se utilizarán cuatro bombillas de 25w, 40w, 60w y 100w. Cada una de ellas, conectada a la corriente, consumirá una potencia

diferente y debido a ello se tendrá cuatro valores de corriente diferentes en el cable, que se podrán medir con el polímetro y con nuestro medidor SCT-013.

Potencia	Medición SCT-013	Medición Polímetro	Conversión polímetro a SCT-013
25W	264 a 267	103.5 mA	2.56 veces
40W	273 a 274	158.4 mA	1.72 veces
60W	445 a 452	230 mA	1.94 veces
100W	661 a 666	365 mA	1.81 veces

Tabla 10: Emontx calibración, mediciones para constante 111.1

Se realiza la misma medición para una constante de calibración de 29.1:

Potencia	Medición SCT-013	Medición Polímetro	Conversión polímetro a SCT-013
25W	67 a 68	103.5 mA	0.64 veces
40W	81 a 82	158.4 mA	0.51 veces
60W	116 a 117	230 mA	0.50 veces
100W	181 a 182	365 mA	0.49 veces

Tabla 11: Emontx calibración, mediciones para constante 29.1

Es posible observar que la variación en la constante de calibración ha supuesto un cambio proporcional en la medición realizada por el SCT-013. El objetivo es igualar la medición realizada por el polímetro y la medición realizada por el SCT-013, con la mayor precisión posible en los diferentes valores de corriente.

Se puede utilizar el siguiente cálculo:

Constante Conocida / Medición Conocida = Constante Adecuada / Medición Deseada

Con dicho cálculo es posible despejar el valor adecuado para la constante de calibración adecuada para cada una de las potencias.
Por ejemplo para una potencia de 100W: 29.1/182 = x/365 x=58.6
Para una potencia de 60W: 29.1/117 = x/230 x=57.2
Para una potencia de 40W: 29.1/82 = x/158 x=56.07

Para una potencia de 25W: 29.1/68 = x/103.5 x=44.29

El único detalle a destacar es que según las mediciones se acercan al mínimo de 50mA que es capaz de medir el SCT-013, el margen de error es mayor. Pero normalmente la aplicación se moverá en mediciones superiores donde se observa una precisión aceptable.

En base a los cálculos se saca una media de las constantes calculadas. Se toma como valor más ajustado para la constante de calibración el valor 56.9. Se vuelve a realizar la misma medición con SCT-013 y polímetro, esta vez para dicha constante de calibración:

Potencia	**Medición SCT-013**	**Medición Polímetro**	**Conversión polímetro a SCT-013**
25W	137 a 138	103.5 mA	1.33 veces
40W	155 a 156	158.4 mA	0.98 veces
60W	228 a 229	230 mA	0.99 veces
100W	369 a 372	365 mA	0.98 veces

Tabla 12: Emontx calibración, mediciones para constante 56.9

Se debe tener en cuenta que siempre existirá un margen de error, pero como se puede observar, existe una precisión aceptable entre las mediciones hechas con el polímetro y las mediciones hechas con el SCT-013. El caso en que se observa un desvío mayor es en valores cercanos al mínimo capaz de medir 50mA, pero a partir de los 150 mA la desviación decrece y se mantiene estable según se aumenta la corriente que atraviesa el cable.

5.3. Detalles de implementación en el centro de control principal

Más allá de lo comentado en el apartado de diseño no es necesario profundizar excesivamente en el desarrollo de la aplicación del centro de control. Esta parte del sistema no ha supuesto grandes problemas de implementación.

La aplicación del centro de control se ejecuta sobre el sistema operativo Raspbian para Raspberry Pi, una versión derivada de Debian.

Cuando se conecta un Arduino por USB al Raspberry Pi, desde el punto de vista de usuario del sistema operativo, aparece un nuevo *device* serie en la ruta /dev/serial/by-id. Este

dispositivo comienza con cadena usb-Arduino__www.arduino.cc__ seguido por un id que identifica al dispositivo concreto.

Desde el punto de vista de la programación, para poder conectar con el Arduino coordinador de la red ZigBee, se debe enumerar los *devices* conectados. Se debe enumerar las entradas de la ruta /dev/serial/by-id, hasta encontrar la de usb-Arduino. Posteriormente se debe abrir el device como si de un fichero se tratara y realizar las lecturas sobre el mismo.

El Arduino coordinador de la red estará reenviando por USB todo lo recibido desde la red ZigBee. Todos los datos reenviados por el coordinador pueden ser leídos en el centro de control a través del *device* mencionado anteriormente como si de un fichero corriente se tratara.

Adicionalmente comentar que se ha utilizado el *kit* de desarrollo para interfaces gráficas Swing para Java. Swing ofrece una API con el que desarrollar interfaces gráficas con componentes más sofisticados a la vez que ofrece un *kit* de diseño y desarrollo que facilita enormemente la implementación de interfaces gráficas en Java. [23] Por ello Swing ha sido elegida como la plataforma idónea para los requisitos gráficos de esta aplicación.

Capítulo 6 Pruebas realizadas

Como ya se ha visto en los apartados anteriores, el sistema de medición de consumo energético que se describe a lo largo de este proyecto está compuesto de varias partes. Durante la fase de desarrollo ha sido necesario dividir el proyecto en cada una de estas partes, desarrollarlas y probarlas por separado asegurando su correcto funcionamiento, para después juntar cada una de ellas para componer el sistema completo.

6.1. Pruebas durante el desarrollo del medidor

Antes de desarrollar el resto de componentes del sistema es necesario asegurar el correcto funcionamiento de los medidores. Tras desarrollar el código que se cargará en cada placa Arduino, se comprobará su funcionamiento.

Inicialmente no será necesario tener conectado el Arduino medidor al módulo Xbee Series 2, solamente a la placa Emontx. Se conectará el Arduino medidor a un equipo ejecutando Microsoft Windows a través de una conexión USB.

Terminado el desarrollo de la aplicación con el IDE Arduino-1.0.5, se cargará el código a la placa. Dicho código enviará las lecturas realizadas o cualquier tipo de respuesta hacia el coordinador a través del puerto serie estándar de Arduino, que, al no estar aún conectado el módulo Xbee Series 2, saldrá por el puerto USB, a su vez conectado al ordenador.

En IDE Arduino-1.0.5 se dispone de la opción Herramientas -> Monitor Serial, mediante la cual es posible ver toda la información que nos llega del Arduino medidor conectado. Mediante esta utilidad es posible enviar peticiones al medidor así como ver todas las respuestas que se reciban.

Durante el desarrollo también es útil añadir trazas al código que se cargará a Arduino para seguir con facilidad la ejecución del programa. Dichas través serán enviadas también por el puerto serie y serán observadas con el monitor serie.

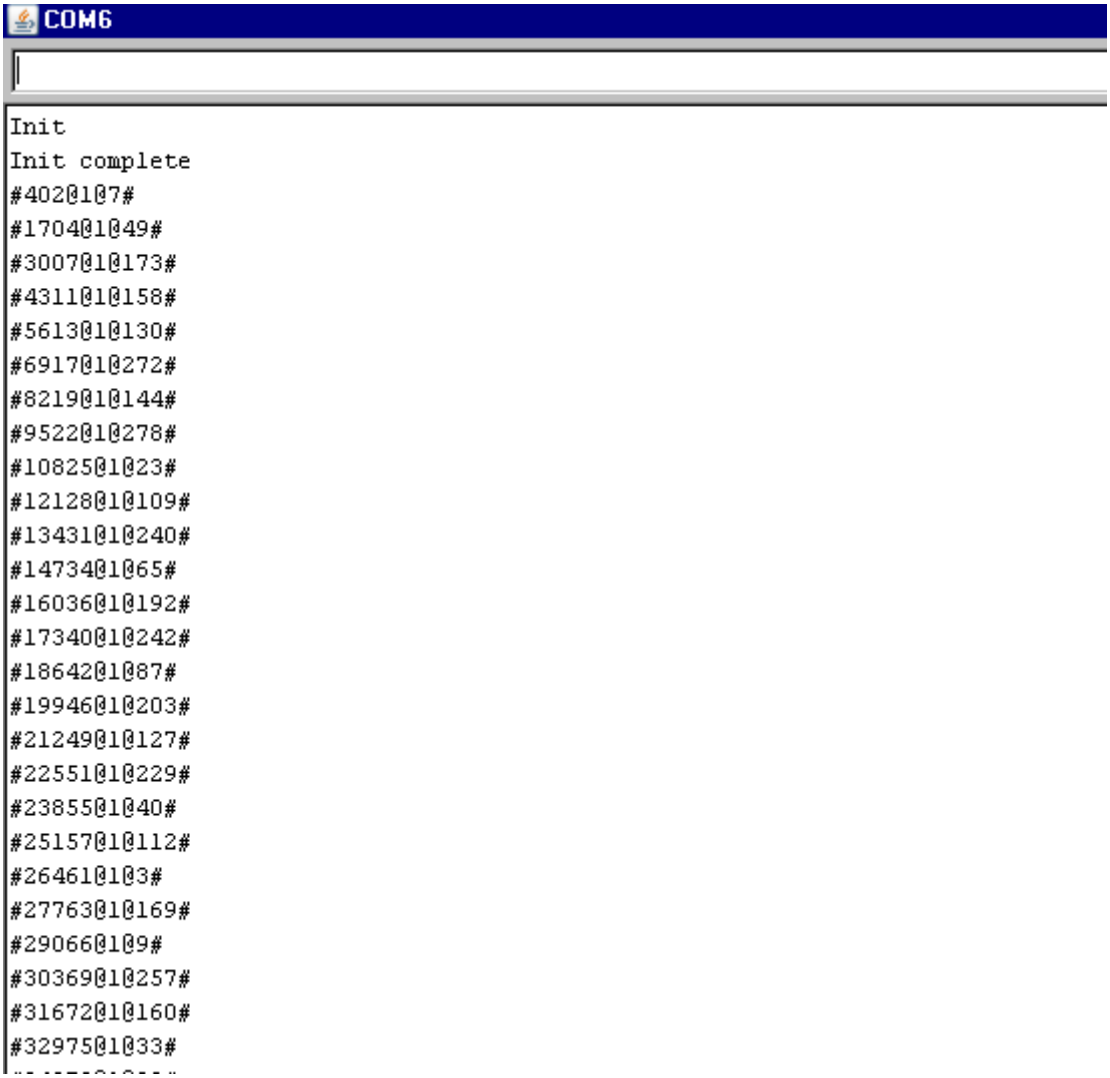

Figura 41: Utilidad monitor serie del IDE Arduino-1.0.5

6.2. Pruebas durante el desarrollo del coordinador

El siguiente paso es desarrollar el coordinador de la red ZigBee. El código del coordinador es muy sencillo, simplemente recibirá datos de la red ZigBee y los reenviará hacia el puerto USB. Por lo tanto el desarrollo del código Java del coordinador no supone mayor problema.

La mayor dificultad, llegados a este punto, es configurar correctamente todos los módulos Xbee Series 2 tal y como se describió en los apartados sobre detalles de implementación.

Una vez realizada dicha configuración se debe dotar a cada Arduino de su módulo de conexión ZigBee correspondiente, conectar el coordinador al puerto USB de un ordenador ejecutando Microsoft Windows, poner en marcha todos los nodos de la red, y una vez más observar mediante el monitor serie de Arduino-1.0.5 que toda la información llega

adecuadamente desde los nodos medidores hasta el coordinador, y desde el coordinador hasta el ordenador a través del cable USB por el que está conectado.

6.3. Pruebas durante el desarrollo del centro de control

Puesto que la aplicación es desarrollada en Java para la plataforma Raspbian, derivada de Debian, y que dicha plataforma distribuye una versión completa y actualizada de Java, es posible desarrollar la parte gráfica del centro de control sobre Microsoft Windows utilizando el entorno de desarrollo Eclipse y aprovechar todas sus comodidades para desarrollo y testeo.

Finalizada la parte gráfica, la parte de comunicaciones con la red ZigBee es necesario testearla directamente sobre el dispositivo Raspberry Pi, al que estará conectado el coordinador por USB. Como ya se dijo una vez conectado el Arduino coordinador aparecerá un nuevo *device* en la Raspberry Pi, en la ruta /dev/serial/by-id.

Para asegurar la correcta conectividad del centro de control con el coordinador de la red ZigBee y de éste con el resto de los dispositivos, es posible ejecutar el comando cat de linux directamente sobre el *device* del Arduino coordinador. De esta manera será posible observar como se reciben las lecturas de la red de un modo muy similar a como se recibían a través del monitor serie del entorno de desarrollo Arduino-1.0.5.

Comprobada la conectividad, el siguiente paso es dejar al centro de control que abra dicho dispositivo y muestre las lecturas a través de su interfaz gráfica.

6.4. Pruebas del sistema completo

Una vez se ha comprobado el correcto funcionamiento de cada parte del sistema por separado, se pasa a comprobar el funcionamiento del sistema completo.

Para ello se utilizará un cable conectado a la red eléctrica de 220V, que suministrará corriente para encender bombillas de cuatro potencias diferentes: 25W, 40W, 60W y 100W. Será necesario también disponer de un polímetro que garantice que las mediciones realizadas por nuestros medidores son correctas.

Se realizarán las mediciones con cada una de las bombillas. Se debe observar que dichas mediciones aparecen correctamente en nuestro centro de control, y que además las mediciones

que aparecen en el centro de control se corresponden en ese momento con las mediciones que muestra el polímetro.

En las pruebas realizadas a continuación se puede ver en rojo las mediciones tomadas por el nodo con id 3, que está conectado a la Emontx tomando medidas reales. Mientras tanto el nodo con id 1 no está conectado a una Emontx y está enviando valores aleatorios.

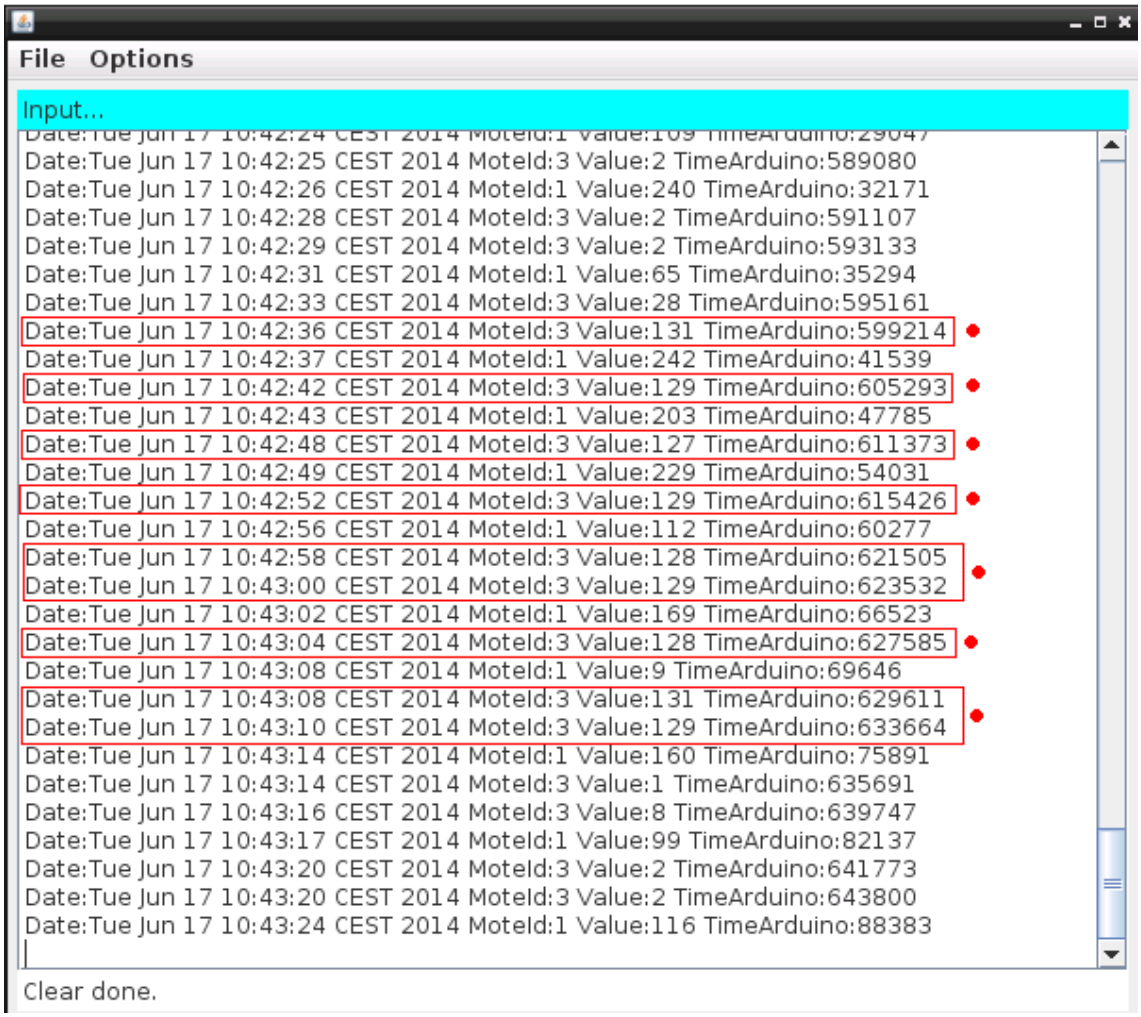

Figura 42: Mediciones utilizando bombilla de 25W con dos nodos en la red

Se puede ver que los valores oscilan entre 127mA y 131mA. La medición del polímetro, sin embargo, es de 105.2 mA. Esta diferencia se debe a que, si se acerca demasiado al mínimo que es capaz de medir el SCT-013, el margen de error aumenta. En valores inferiores a 120 mA, el margen de error es algo mayor. A partir de 150 mA en adelante, el margen de error disminuye y en adelante se mantiene estable.

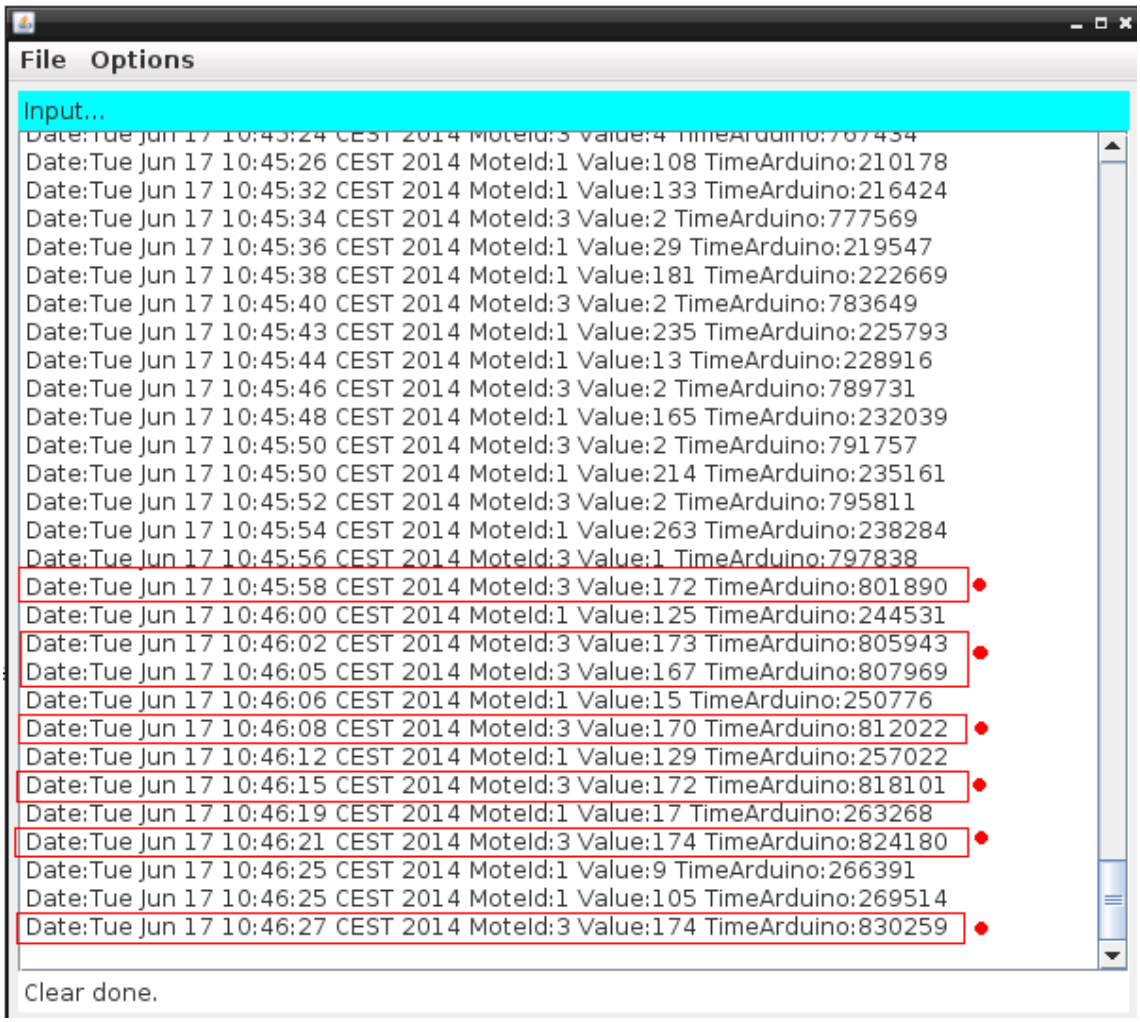

Figura 43: Mediciones utilizando bombilla de 40W con dos nodos en la red

Para la bombilla de 40W el polímetro marca 158.4 mA. Nuestras mediciones están en torno a 167mA.

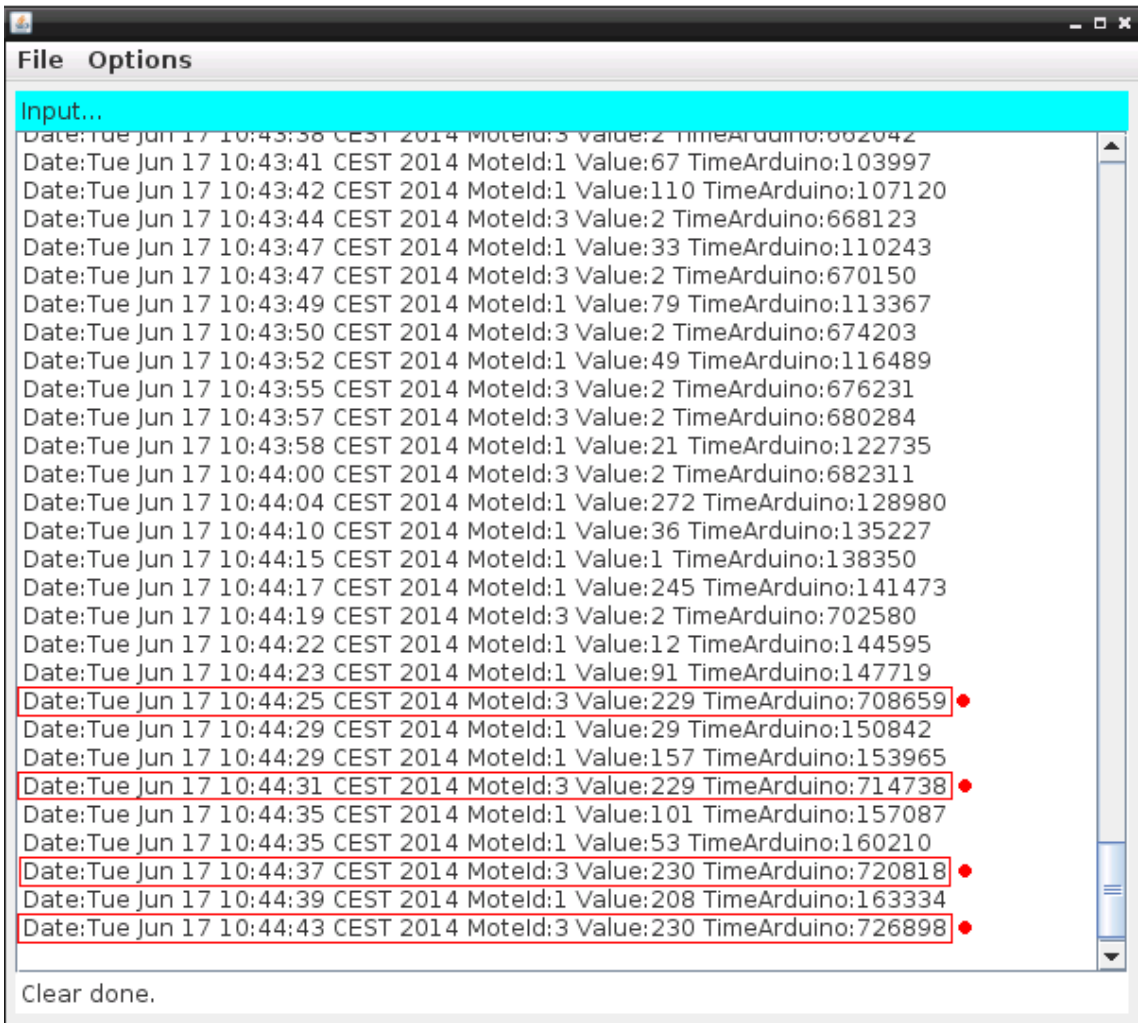

Figura 44: Mediciones utilizando bombilla de 60W con dos nodos en la red

La medida del polímetro para el caso de la bombilla de 60W es de 230 mA. En la captura se ve que las medidas del SCT-013 oscilaban en torno a 229 mA y 230 mA. Se puede observar como a mayores valores de corriente el SCT-013 mide con mayor precisión.

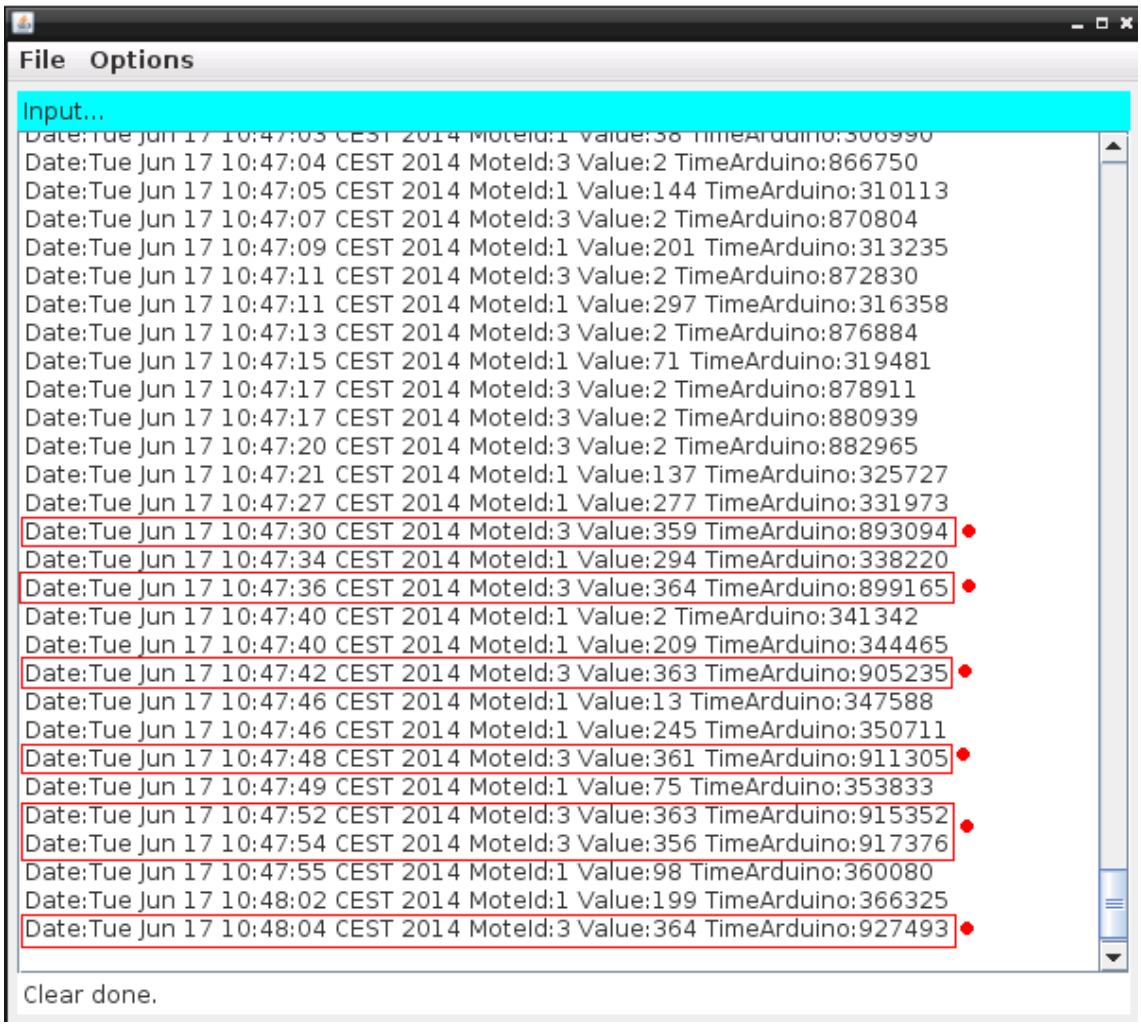

Figura 45: Mediciones utilizando bombilla de 100W con dos nodos en la red

En este último caso se observa que las medidas giran en torno a 359 mA hasta 364 mA. El polímetro mide 365 mA en este caso, por lo que la medición es bastante aproximada.

Capítulo 7 Conclusiones

En este proyecto de fin de carrera se ha hecho un estudio acerca del estado del arte en cuanto a gestión de la información en redes de sensores inalámbricos. Se ha escogido la plataforma TinyDB como elemento de estudio, por ser posiblemente el sistema que ofrece un mayor grado de abstracción en cuanto a manejo de redes de sensores y gestión de la información en las mismas.

TinyDB ofrece una visión de la red de sensores como si de una base de datos se tratara. Una base de datos sobre la que se pueden realizar consultas como en cualquier base de datos, aunque con sus peculiaridades. Además TinyDB implementa multitud de optimizaciones en cuanto a manejo de datos, reducción de tráfico de red y minimización en consumo energético por parte de los nodos de la red de sensores. En este proyecto se han analizado en profundidad todos estos conceptos, ideas y optimizaciones implementados por TinyDB.

Tras el estudio teórico de las redes de sensores inalámbricas y de la plataforma TinyDB, como ejemplo del estado del arte en gestión de la información de este tipo de redes, se ha pasado a la parte práctica del proyecto, en la cual se ha desarrollado un sistema de monitorización de consumo energético basado en redes de sensores. Este sistema se descompone en dos partes principales, una red de sensores inalámbricos, extractora de datos, y un centro de control, que maneja la información recibida de la red. La red de sensores podría a su vez descomponerse en dos tipos de nodos, los nodos terminales de la red (aquellos con capacidad para realizar mediciones de consumo) y los nodos coordinadores de la red (solamente existe uno por cada red y se encarga de conectar el centro de control con la red de sensores).

Se han elegido las plataformas hardware Arduino para cada nodo de la red de sensores. A su vez se ha dotado a cada placa Arduino con un módulo Xbee conectable, con la finalidad de poder crear la red de sensores basada en dispositivos Arduino. Y por último, se ha utilizado la placa Emontx y los sensores de corriente eléctrica SCT-013, para dar a cada nodo terminal la capacidad de realizar mediciones de consumo eléctrico. En cuanto al centro de control, se ha utilizado la plataforma Raspberry Pi, un ordenador empotrado que ejecuta el sistema operativo de propósito general Raspbian (una versión derivada de Debian) y que tiene un potencial mucho mayor que el de los Arduino. Sobre Raspberry Pi se ha implementado un sencillo centro de control que gestiona el conocimiento recibido de la red, y si fuera necesario, se podría ampliar para crear un centro de control más avanzado que, por ejemplo, pudiera enviar el conocimiento recibido de la red a un servidor a través de Internet.

Se ha escogido el lenguaje de programación Java, tanto para crear la aplicación que ejecutarán los nodos terminales como para crear la aplicación que ejecutará el coordinador y el centro de

control. La plataforma Arduino es programable en Java (si bien con algunas limitaciones en cuanto a API disponibles), y la plataforma Raspbian distribuye una versión completa y actualizada de la máquina virtual de Java, por lo tanto Java es el lenguaje idóneo a utilizar durante el desarrollo del sistema, para mantener una coherencia entre las distintas partes.

Como conclusión se puede decir que se han cumplido los objetivos planteados para este proyecto: se ha realizado un estudio sobre el estado del arte en gestión de la información en redes de sensores inalámbricos, y se ha desarrollado un sistema útil en distintos ámbitos como en la industria o en los hogares.

7.1 Trabajos futuros

En este proyecto se ha realizado un estudio de la plataforma TinyDB y se han recopilado ideas que posteriormente se han utilizado en el desarrollo de un sistema de monitorización de consumo energético para entornos industriales y domésticos. TinyDB existe para la plataforma TinyOS, pero para el desarrollo del sistema de monitorización de consumo se ha utilizado la plataforma Arduino por ser una plataforma potente e idónea para los objetivos marcados, como se ha comentado en el capítulo sobre elección de hardware a utilizar. A pesar de no haber utilizado TinyDB, se han implementado en el sistema desarrollado múltiples optimizaciones inspiradas en dicha plataforma. Sin embargo sería interesante como trabajo para el futuro desarrollar un sistema similar esta vez utilizando la plataforma TinyDB, sobre motas que ejecuten TinyOS.

Como se ha comentado en otros apartados, se ha escogido la plataforma Raspberry Pi para desarrollo del centro de control del sistema de monitorización de consumo. Esta potente plataforma permitiría ampliar enormemente la capacidad de dicho centro de control. Sería un buen punto sobre el que mejorar como trabajo futuro la ampliación de dicho centro de control para, por ejemplo, enviar los datos a un servidor remoto o permitir el acceso remoto al centro de control, de manera que se pudieran controlar a distancia múltiples instalaciones del sistema de monitorización de consumo.

Por último cabe destacar que el sistema desarrollado es capaz de monitorizar el consumo energético en el entorno donde esté instalado. Sin embargo, y aprovechando que se ha desplegado una red de sensores inalámbricos sobre una zona, sería mucho más óptimo dotar a los dispositivos medidores de más tipos de sensores (consumo energético, temperatura, luz, humedad, detectores de gas u otros compuestos, etc…) de manera que se tuviera un sistema de propósito general mucho más inteligente que monitorizara diferentes parámetros de la industria o el hogar.

Capítulo 8 Bibliografía

[1]Jason Lester Hill, (2003), System Architecture for Wireless Sensor Networks. [En línea]. Disponible: http://www.eps2009.dj-inod.com/docs/09-02-01/system_architecture_for_wireless_sensor_networks.pdf

[2]Synny Liu, Mary Parmelee, (2002), Databases and Knowledge Management. [En línea]. Disponible: http://www.unc.edu/~sunnyliu/inls258/

[3]Wikipedia, The Free Encyclopedia, (2014, Junio 4), Arduino. [En línea]. Disponible: http://en.wikipedia.org/wiki/Arduino

[4]Wikipedia, The Free Encyclopedia, (2014, Junio 9), Raspberry Pi. [En línea]. Disponible: http://en.wikipedia.org/wiki/Raspberry_Pi

[5]Greg Hackmann, (2006, Marzo 21), 802.15 Personal Area Networks. [En línea]. Disponible: http://www.cse.wustl.edu/~jain/cse574-06/ftp/wpans.pdf

[6]Dusan Stevanovic, (2007, Junio), Standard IEEE 802.15. [En línea]. Disponible: http://www.cse.yorku.ca/~dusan/Zigbee-Standard-Talk.pdf

[7]Wikipedia, The Free Encyclopedia, (2014, Junio 12), Standard Zigbee. [En línea]. Disponible: http://en.wikipedia.org/wiki/ZigBee

[8]ZigBee Alliance, (2008, Enero), Zigbee Specification. [En línea]. Disponible: http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/kjb79_ajm232/pmeter/ZigBee%20Specification.pdf

[9]Sam Madden, Joe Hellerstein, and Wei Hong, (2003, Septiembre), TinyDB: In-Network Query Processing in TinyOS. [En línea]. Disponible: http://telegraph.cs.berkeley.edu/tinydb/tinydb.pdf

[10]TinyOS Community, (2002, Septiembre), TinyDB A Declarative Query System for Motes. [En línea]. Disponible: http://www.tinyos.net/tinyos-1.x/doc/tutorial/tinydb.html

[11]Arduino Community, (2014, Junio), Arduino Store. [En línea]. Disponible: http://store.arduino.cc/

[12]Arduino Community, (2014, Junio), Arduino Xbee Shield. [En línea]. Disponible: http://arduino.cc/en/Main/ArduinoXbeeShield

[13]Arduino Community, (2014, Junio), Arduino Schematics. [En línea]. Disponible: http://www.arduino.cc/en/uploads/Main/XbeeShieldSchematic.pdf

[14]Arduino Community, (2014), Arduino Wireless Shield with XBee Series 2 radios, Junio. [En línea]. Disponible: http://arduino.cc/en/Guide/ArduinoWirelessShieldS2

[15]Digi.com Community, (2014, Junio), Xbee/Xbee-PRO ZB RF Modules Datasheet. [En línea]. Disponible: http://ftp1.digi.com/support/documentation/90000976_S.pdf

[16]Arduino Community, (2014, Junio), Arduino Java Language Reference. [En línea]. Disponible: http://arduino.cc/en/Reference/HomePage

[17]Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, (2004), TinyDB: An Acquisitional Query Processing System for Sensor Networks. [En línea]. Disponible: http://db.cs.berkeley.edu/papers/tods05-tinydb.pdf

[18]Digi.com Community, (2014), Next generation configuration platform for XBee. [En línea]. Disponible: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu

[19]Donal Morrisey, (2011), Sleeping Arduino - Wake Up Via The Watchdog Timer. [En línea]. Disponible: http://donalmorrissey.blogspot.com.es/2010/04/sleeping-arduino-part-5-wake-up-via.html

[20]Tiker London Community, (2009), Accessing the secret voltmeter on the Arduino 168 or 328. [En línea]. Disponible: https://code.google.com/p/tinkerit/wiki/SecretVoltmeter

[21]OpenEnergyMonitor Community, (2014), Emontx Calibration. [En línea]. Disponible: http://openenergymonitor.org/emon/buildingblocks/calibration

[22]OpenEnergyMonitor Community, (2014), Writting Code For The Emontx. [En línea]. Disponible: http://openenergymonitor.org/emon/emontx/firmware/writing-code-for-the-emontx

[23]Wikipedia, The Free Encyclopedia, (2010), Swing. [En línea]. Disponible: http://en.wikipedia.org/wiki/Swing_(Java)

[24]OpenEnergyMonitor Community, (2014), Emontx: Installation and Calibration. [En línea]. Disponible: http://openenergymonitor.org/emon/buildingblocks/ct-and-ac-power-adaptor-installation-and-calibration-theory